



Universal Data Mover

User Guide

Universal Products

Version 3.2.0

Universal Data Mover

User Guide

Universal Products 3.2.0

Document Name	Universal Data Mover 3.2.0 User Guide				
Document ID	udm-user-3207				
Products	z/OS	UNIX	Windows	OS/400	HP NonStop
Universal Data Mover Manager	✓	✓	✓	✓	
Universal Data Mover Server	✓	✓	✓	✓	

Stonebranch Documentation Policy

This document contains proprietary information that is protected by copyright. All rights reserved. No part of this publication may be reproduced, transmitted or translated in any form or language or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission, in writing, from the publisher. Requests for permission to make copies of any part of this publication should be mailed to:

Stonebranch, Inc.
950 North Point Parkway, Suite 200
Alpharetta, GA 30005 USA
Tel: (678) 366-7887
Fax: (678) 366-7717

Stonebranch, Inc.® makes no warranty, express or implied, of any kind whatsoever, including any warranty of merchantability or fitness for a particular purpose or use.

The information in this documentation is subject to change without notice.

Stonebranch shall not be liable for any errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this document.

All products mentioned herein are or may be trademarks of their respective owners.

© 2003-2010 by Stonebranch, Inc.

All rights reserved.



Summary of Changes

Changes for Universal Data Mover 3.2.0 User Guide (udm-user-3207) February 19, 2010

- Added `_execrc` built-in variable in Section [11.6.8 Built-in Variables](#).

Changes for Universal Data Mover 3.2.0 User Guide (udm-user-3206) September 8, 2009

- Moved Universal Data Mover Manager examples for all operating systems to [Appendix A Examples](#).

Universal Data Mover 3.2.0.6

- Added LOGON_METHOD configuration option in [Chapter 8 Universal Data Mover Server for Windows](#).

Changes for Universal Data Mover 3.2.0 User Guide (udm-user-3205) July 29, 2009

Universal Data Mover 3.2.0.1 for OS/400

- Modified document for upgrade from Universal Data Mover 3.1.1 for OS/400 to Universal Data Mover 3.2.0 for OS/400, including:
 - Changed the following OS/400 names throughout the document:
 - Universal Broker subsystem name from **UBROKER** to **UNVUBR320**.
 - Universal Broker user profile name from **UBROKER** to **UNVUBR320**.
 - Universal Products installation library name from **UNIVERSAL** to **UNVPRD320**.

- Universal Products spool library name from **UNVSP00L** to **UNVSPL320**.
- Universal Products temporary directory from **UNVTMP** to **UNVTMP320**.
- Added Section [6.2.1 Universal Products for OS/400 Commands](#).
- Added the following configuration options in Section [6.2.4 Configuration Options of Chapter 6 Universal Data Mover Manager for OS/400](#):
 - ACTIVITY_MONITORING
 - CA_CERTIFICATES
 - CERTIFICATE
 - CERTIFICATE_REVOCATION_LIST
 - CODEPAGE_TO_CCSDID_MAP
 - COMMENT
 - EVENT_GENERATION
 - OPEN_RETRY
 - OPEN_RETRY_COUNT
 - OPEN_RETRY_INTERVAL
 - PLF_DIRECTORY
 - PRIVATE_KEY
 - PRIVATE_KEY_PWD
 - PROXY_CERTIFICATES
- Added the following STRUDM parameters in [Figure 6.1 UDM Manager for OS/400 - Command Line Syntax](#):
 - CTLCPHRLST
 - DTACPHRLST
 - FRAMEINT
 - MODETYPE
 - MSGLEVEL (time values)
 - OUTBOUNDIP
- Added an [Invoke a Script from a Batch Job](#) example in Appendix [A.4 UDM Manager for OS/400 Examples](#).
- Added the following configuration options in Section [10.3.2 Configuration Options of Chapter 10 Universal Data Mover Server for OS/400](#):
 - ACTIVITY_MONITORING
 - CODEPAGE_TO_CCSDID_MAP
 - EVENT_GENERATION
 - TMP_DIRECTORY
- Modified Section [14.4.2 call \(Call\) Command](#) in [Chapter 14 Transfer Operations \(OS/400-Specific\)](#).
- Added [Caution about Text Mode Transfer of Files with DDS](#) in subsection [14.2.4 Data Physical Files Support](#) of Section [14.2 OS/400 I/O](#).

Changes for Universal Data Mover 3.2.0 User Guide

(udm-user-3204)

April 1, 2009

- Moved the Licenses and Copyrights appendix to the Universal Products 3.2.0 Installation Guide.

Universal Data Mover 3.2.0.3

- Added the TCP_NO_DELAY configuration option to the following tables:
 - [Chapter 3 Universal Data Mover Manager for z/OS](#)
 - [Chapter 4 Universal Data Mover Manager for Windows](#)
 - [Chapter 5 Universal Data Mover Manager for UNIX](#)
 - [Chapter 7 Universal Data Mover Server for z/OS](#)
 - [Chapter 8 Universal Data Mover Server for Windows](#)
 - [Chapter 9 Universal Data Mover Server for UNIX](#)
- Added the following commands in [Table 11.1 UDM Commands](#):
 - appenddata
 - closelog
 - echolog
 - logdata
 - move
 - openlog
 - savedata
- Added [Table 11.4 _file Built-in Variable – Special Attributes](#).
- Added the following built-in variables to Section [11.6.8 Built-in Variables](#):
 - [_uuid](#)
 - [_lastmsg](#)
- Added [sortby](#) parameter information in Section [11.10 forfiles Statement](#).
- Added [_file Variable Attributes](#) section in Section [11.10.1 forfiles Built-In Variables](#).
- Added Section [12.8.2 Move Operation](#).
- Added the following parameters in Section [15.2 exec Command](#):
 - stdout
 - stderr

Changes for Universal Data Mover 3.2.0 User Guide

(udm-ref-3203)

December 17, 2008

- Added [Updating the Universal Data Mover Server ACL Entries](#) in Section [8.4.5 Universal Access Control List](#) of Chapter 8 Universal Data Mover Server for Windows.

Changes for Universal Data Mover 3.2.0 User Guide (udm-ref-3202) October 17, 2008

- Changed JCL SNTYPE value to `ttype` for the `dsntype` attribute in [Table 13.1 attrib Command - Dynamic Allocation Attributes](#).

Changes for Universal Data Mover 3.2.0 User Guide (udm-user-3201) September 5, 2008

- Added toll-free telephone number for North America in [Appendix B Customer Support](#).

Changes for Universal Data Mover 3.2.0 User Guide (udm-user-320) May 16, 2008

Universal Data Mover 3.2.0.0

- Added support for the following features:
 - Script Language
UDM has made vast improvements in the power of its scripting language with the addition of new control structures and built in functions.
 - X.509 Certificates
UDM provides full support for X.509 certificates.
 - UNIX Permissions
When transferring UNIX files between two UNIX systems, the file permission modes of the destination files may now be based upon the source file modes.
 - z/OS Load Module Copies
UDM on z/OS now supports the transfer of load modules and program objects between z/OS systems.
 - z/OS Relative GDG Delete
Support for deleting z/OS generation data sets based upon a relative number using the UDM DELETE command.
 - z/OS GDG Resolution Method
The method used for resolving for generation data set relative numbers with a UDM script can be selected between the original job method and the new reference method.
 - Manager Identification in Three-Party Transfers
The UDM manager identity, either user identifier and IP address or X.509 certificate information, is propagated to the primary and secondary UDM servers and available for UACL processing.
- Deleted the following values in [Table 2.3 Component Communication States](#):
 - ORPHANED

- PENDING
- RESTARTING
- Added [Chapter 2 Features](#), including:
 - Section [2.4 Universal Configuration Manager](#).
 - Section [2.6.2 Open Retry](#).
 - Section [2.7 z/OS CANCEL Command Support](#).
 - Section [2.8.3 Types of UACL Rules](#).
 - Section [2.8.4 Proxy Certificates](#).
- Added the following UDM Manager configuration options for the z/OS, Windows, and UNIX operating systems:
 - ACTIVITY_MONITORING
 - BIF_DIRECTORY (UNIX only)
 - CA_CERTIFICATES
 - CERTIFICATE
 - CERTIFICATE_REVOCATION_LIST
 - COMMENT
 - EVENT_GENERATION
 - OPEN_RETRY_COUNT
 - OPEN_RETRY_INTERVAL
 - PLF_DIRECTORY (UNIX only)
 - PRIVATE_KEY
 - PRIVATE_KEY_PASSWORD
 - PROXY_CERTIFICATES
 - SAF_KEY_RING (z/OS only)
 - SAF_KEY_RING_LABEL (z/OS only)
 - SERVER_STOP_CONDITIONS (z/OS only)
 - SSL_IMPLEMENTATION (z/OS only)
 - SYSTEM_ID (z/OS only)
 - UCMD_PATH (UNIX and Windows only)
- Added UDM_MGR_ACCESS UACL entry for the z/OS, Windows, and UNIX operating systems.
- Added the following UDM Server configuration options for the z/OS, Windows, and UNIX operating systems:
 - ACTIVITY_MONITORING
 - EVENT_GENERATION
 - TMP_DIRECTORY
- Added [Nesting / Recursion of Subroutines](#) in Section [11.5 Subroutines](#) of [Chapter 11 UDM Scripting Language](#).
- Changed the `_result` variable to an attribute of the `_lastrc` variable in Section [11.6.8 Built-in Variables](#) of [Chapter 11 UDM Scripting Language](#).
- Added information to [Chapter 11 UDM Scripting Language](#):
- Added and modified information in [Chapter 16 Return Code Processing](#):
- Added Section [12.8.6 File Permission Attribute](#).
- Added Section [13.4 Copying Load Modules](#).

- Consolidated UDM Manager Configuration chapters and UDM Manager Invocation chapters.

Changes for Universal Data Mover 3.1.1 User Guide (udm-user-31111) February 28, 2007

- Added customer support telephone number for Europe to Appendix B Customer Support.
- Universal Data Mover 3.1.1.7
- Added svropt option to the following sections:
 - Section 11.1 exec Command in Section 11 Remote Execution.
 - Section 13.10 exec in Section 13 Command Reference.
- Added paragraph about HSM data migration to Section 9.1.4 Allocation of Section 9 z/OS-Specific Transfer Operations.
- Added ASPDEV value for ASPNUM attribute in Table 10 OS/400-Specific LIB File Attributes for Creating New Files in Section 10.2.1 attrib (Attribute) Command of Section 10 OS/400-Specific Transfer Operations
- Updated network fault tolerance information for the NETWORK_FAULT_TOLERANT option in the following sections:
 - Section 3.1 Configuring the UDM Manager of Section 3 Configuring UDM under UNIX
 - Section 4.1 Configuring UDM Manager of Section 4 Configuring UDM under z/OS
 - Section 5.1 Configuring UDM Manager of Section 5 Configuring UDM under OS/400
 - Section 6.1 Invoking UDM Manager under Windows and UNIX
 - Section 6.2 Invoking UDM Manager Under z/OS
 - Section 6.3 Invoking UDM Manager Under OS/400
- Added the OPEN_RETRY UDM Manager configuration option in the following sections:
 - Section 3.1 Configuring UDM Manager of Section 3 Configuring UDM under UNIX.
 - Section 4.1 Configuring UDM Manager of Section 4 Configuring UDM under z/OS.
 - Section 6.1 Invoking the UDM Manager under Windows and UNIX
 - Section 6.2 Invoking the UDM Manager Under z/OS

Changes for Universal Data Mover 3.1.1 User Guide (udm-user-31110) December 15, 2006

- Added List of Figures and List of Tables.
- Added Customer Support.
- Universal Data Mover 3.1.1.6
- Added Section 1.4 Configuration File or System Registry.

- Added Section 5 Configuring UDM under OS/400.
- Added Section 6.3 Invoking the UDM Manager Under OS/400.
- Added Section 8.5 OS/400 File Systems.
- Added Section 10 OS/400-Specific Transfer Operations.
- Changed name of Configuration File Keyword from receive_buffer_size to recv_buffer_size (throughout document)
- Added default values for options in Section 2 Configuring UDM under Windows.
- Added Section 2.3.1 Component Definitions to Section 2.3 Configuring Universal Data Mover Server.
- Added the MERGE_LOG option to the following sections:
 - 3.1 Configuring the UDM Manager of Section 3 Configuring UDM under UNIX.
 - 4.1 Configuring the UDM Manager of Section 4 Configuring UDM under z/OS.
 - 6.1 Invoking the UDM Manager under Windows and UNIX of Section 6 UDM Manager Invocation.
 - 6.2 Invoking the UDM Manager Under z/OS of Section 6 UDM Manager Invocation.
- Added default value to USER_SECURITY in Section 3.2.2.2 Configuration Options Reference.
- Added an "If" statement to the sample UDM script in Section 7.2 UDM Command Format.
- Redefined Section 7.3 Script Files, and renamed and redefined Sections 7.3.1 Invoking UDM in Batch with Commands from a Script File and 7.3.2 Invoking UDM Interactively with Commands from a Script File.
- Added LIB as a value for the filesys command, and Indicated availability of filesys value of DD, in the following sections:
 - Section 8.2.2 Changing the Current File System of Section 8 UDM Transfer Operations
 - Section 13.13 filesys of Section 13 Command Reference
- Added information about EOL attribute in HFS file system in Section 8.6.3 End of Line Sequence.
- Added note in Section 8.7.6 Transaction-Oriented Transfers about non-support in UDM OS/400 LIB file system.
- Added Syntax, Description, and Parameters for Section 13.11 execsap in Section 13 Command Reference.
- Added default value for logical-name parameter in Section 13.13 filesys.
- Modified example in Section 14.1.5 Copy a Set of Files to a New z/OS Partitioned Data Set.
- Added Section 14.3 OS/400 Examples to Section 14 Samples.

Changes for UDM Release 3.1.1.0 April 30, 2005

- Added information about the EXEC command.
- Added information about the EXECSAP command.

- Added information about support for wildcards in data set names for the DSN file system.
- Added information about the DATA command for in-stream data.
- Added the RENAME command.
- Added information about source allocation attributes being used in MVS to MVS copy operations.
- Added information about the _DATE and _TIME built-in variables.
- Added documentation of the on-the-fly logical name built-in variables.
- Documentation for the new TYPE variable attribute for the _FILE built-in variable.
- Updated information on configuration options.
- Added information about the REPORT command.
- Added information about transfer operation auditing.
- Added information about monitoring transfer operation progress.

Changes for UDM Release 3.1.0 October 31, 2004

- Removed section on the control session cipher list configuration as this information now comes from the version 3.1 of the Universal Broker.
- Added information about keep-alive support.
- Added information about console idle timeout.
- Added information about network fault tolerance support in UDM.
- Added information on nested script parameter scope.
- Added z/OS UDM Server.
- Added commands break, copydir and delete.
- Added scripting features of 'IF' logic and 'forfiles'.
- Added section on available built-in variables and reserved variable names.

Changes for UDM Release 1.1.0 February 17, 2004 Release

- Added control and data session cipher list configuration to UNIX UDM Server configuration section.
- Added information about the NULL-NULL cipher to the documentation for the OPEN command.
- Added information about the NULL-NULL cipher for the data session in the UDM Server configuration sections.
- Updated UDM Manager configuration sections with information about setting the default transfer mode type.
- Updated information about default port in the UDM Manager configuration sections.
- Added information about a message timestamp to the MESSAGE_LEVEL configuration option for UNIX and z/OS versions of UDM.
- Added documentation on the resetattrib command.

Contents

Summary of Changes	5
Contents	13
List of Figures	25
List of Tables	27
Preface	30
Document Structure	30
Format	30
Conventions	31
Vendor References	32
Document Organization	33
Chapter 1 Overview	34
1.1 Introduction to Universal Data Mover	34
1.2 Transfer Components	35
1.2.1 Manager	35
1.2.2 Primary Server	35
1.2.3 Secondary Server	35
1.3 Transfer Sessions	36
1.3.1 Logical Names	36
1.3.2 Two-Party Transfer Sessions	36
1.3.3 Three-Party Transfer Sessions	36

Chapter 2 Features	38
2.1 Overview	38
2.2 Configuration	39
2.2.1 Configuration Methods	39
2.2.2 Command Line	40
2.2.3 Command Line File	42
2.2.4 Environment Variables	43
2.2.5 Configuration File	45
2.2.6 Configuration File Syntax	47
2.3 Remote Configuration	48
2.3.1 Unmanaged Mode	48
2.3.2 Managed Mode	49
Selecting Managed Mode	49
2.3.3 Universal Broker Startup	51
2.4 Universal Configuration Manager	52
2.4.1 Availability	52
2.4.2 Accessing the Universal Configuration Manager	54
2.4.3 Navigating through Universal Configuration Manager	56
2.4.4 Modifying / Entering Data	56
Rules for Modifying / Entering Data	56
2.4.5 Saving Data	57
2.4.6 Accessing Help Information	57
2.4.7 Universal Data Mover Installed Components	58
Universal Data Mover Manager	58
Universal Data Mover Server	59
2.5 Network Data Transmission	60
2.5.1 Secure Socket Layer Protocol	60
Data Privacy and Integrity	61
Peer Authentication	62
2.5.2 Universal Products Protocol	63
Data Privacy and Integrity	63
2.5.3 Universal Products Application Protocol	64
Low-Overhead	64
Secure	64
Extensible	65
2.5.4 Configurable Attributes	66
2.6 Fault Tolerance	70
2.6.1 Network Fault Tolerance	70
2.6.2 Open Retry	71
2.6.3 Component Management	72
2.7 z/OS CANCEL Command Support	73
2.7.1 Exit Codes	73

2.7.2 Security Token	74
2.8 Universal Access Control List	75
2.8.1 UACL Configuration	76
2.8.2 UACL Entries	77
Client Identification	77
Certificate-Based and Non Certificate-Based UACL Entries	81
2.8.3 Types of UACL Rules	82
udm_access	82
udm_mgr_access	82
udm_cert_access	82
2.8.4 Proxy Certificates	83
2.9 Message and Audit Facilities	84
2.9.1 Message Types	84
2.9.2 Message ID	85
2.9.3 Message Levels	85
2.9.4 Message Destinations	86
2.10 X.509 Certificates	88
2.10.1 Sample Certificate Directory	89
2.10.2 Sample X.509 Certificate	90
Certificate Fields	91
2.10.3 SSL Peer Authentication	92
Certificate Verification	92
Certificate Revocation	92
Certificate Identification	93
Certificate Support	93
Chapter 3 Universal Data Mover Manager for z/OS	94
3.1 Overview	94
3.2 Usage	95
3.2.1 JCL Procedure	96
3.2.2 DD Statements in JCL	97
3.2.3 JCL	98
3.2.4 Configuration	99
3.2.5 Configuration Options	100
3.2.6 Command Line Syntax	103
3.3 Examples of UDM Manager for z/OS	105
3.4 Security	106
3.4.1 Data Set Permissions	106
Chapter 4 Universal Data Mover Manager for Windows	107
4.1 Overview	107

4.2 Usage	108
4.2.1 Modes of Operation	108
Running UDM in Interactive Mode	108
Running UDM in Batch Mode	108
4.2.2 Configuration	109
4.2.3 Configuration Options	110
4.2.4 Command Line Syntax	112
4.3 Examples of UDM Manager for Windows	113
Chapter 5 Universal Data Mover Manager for UNIX	114
5.1 Overview	114
5.2 Usage	115
5.2.1 Modes of Operation	115
Running UDM in Interactive Mode	115
Running UDM in Batch Mode	115
5.2.2 Configuration	116
5.2.3 Configuration Options	117
5.2.4 Command Line Syntax	119
5.3 Examples of UDM Manager for UNIX	120
5.4 Security	121
5.4.1 File Permissions	121
5.4.2 Configuration Files	121
Chapter 6 Universal Data Mover Manager for OS/400	122
6.1 Overview	122
6.2 Usage	123
6.2.1 Universal Products for OS/400 Commands	123
6.2.2 Modes of Operation	124
Running UDM Interactively	124
Running UDM from a Script	124
Running UDM in Batch Mode	125
6.2.3 Configuration	126
6.2.4 Configuration Options	127
6.2.5 Command Line Syntax	129
6.3 Examples of UDM Manager for OS/400	130
6.4 Security	131
6.4.1 Object Permissions	131
Chapter 7 Universal Data Mover Server for z/OS	132
7.1 Overview	132

7.2 Component Definition	133
7.3 Configuration	134
7.3.1 Configuration File	134
7.3.2 Configuration Options	135
7.4 Security	137
7.4.1 File Permissions	137
7.4.2 Configuration Files	137
7.4.3 Universal Data Mover Server User ID	137
7.4.4 User Authentication	137
7.4.5 Universal Access Control List	138
UACL Entries	138
UACL Examples	139
Chapter 8 Universal Data Mover Server for Windows	140
8.1 Overview	140
8.2 Component Definition	141
8.3 Configuration	143
8.3.1 Configuration File	143
8.3.2 Configuration Options	144
8.4 Security	145
8.4.1 File Permissions	145
8.4.2 Configuration Files	145
8.4.3 Universal Data Mover Server User ID	145
8.4.4 User Authentication	146
8.4.5 Universal Access Control List	146
UACL Entries	146
Updating the Universal Data Mover Server ACL Entries	147
Chapter 9 Universal Data Mover Server for UNIX	148
9.1 Overview	148
9.2 Component Definition	149
9.3 Configuration	150
9.3.1 Configuration File	150
9.3.2 Configuration Options	151
9.4 Security	152
9.4.1 File Permissions	152
9.4.2 Configuration Files	152
9.4.3 Universal Data Mover Server User ID	152
9.4.4 User Authentication	153
9.4.5 Universal Access Control List	153
UACL Entries	153

UACL Examples	154
Chapter 10 Universal Data Mover Server for OS/400	155
10.1 Overview	155
10.2 Component Definition	156
10.3 Configuration	157
10.3.1 Configuration File	157
10.3.2 Configuration Options	158
10.4 Security	159
10.4.1 Object Permissions	159
10.4.2 Universal Data Mover Server User Profile	159
10.4.3 User Authentication	160
10.4.4 Universal Access Control List	160
UACL Entries	160
UACL Examples	161
Chapter 11 UDM Scripting Language	162
11.1 Overview	162
11.2 UDM Commands	163
11.3 UDM Command Format	165
11.3.1 Basic Rules	165
Parameters	165
Spaces	165
Escape Sequences	165
Line Continuation	166
Comments	166
11.3.2 Sample UDM Script	166
11.3.3 Expressions	167
Appearance	167
Integer Only	167
Delimiters	167
Operand / Operator Delimiters	167
Operator Precedence	168
Nesting	168
Operations	169
11.3.4 Strings in Expressions	170
Index Position and Sequence	170
11.3.5 Examples of Expressions	171
11.4 Script Files	172
11.4.1 Invoking UDM in Batch Mode with Commands from a Script File	172
11.4.2 Invoking UDM Interactively with Commands from a Script File	173

11.4.3 Invoking Scripts from within Scripts	174
11.4.4 Parameter Processing	174
11.5 Subroutines	175
11.5.1 Usage	175
Defining a Subroutine	175
Invoking a Subroutine	175
Sequence of Defining / Invoking a Subroutine	176
Nesting / Recursion of Subroutines	176
11.5.2 Example	177
Output	177
11.6 UDM Variables	178
11.6.1 Variable Types	178
Variable Names	178
11.6.2 Variable Reference	178
11.6.3 Script Variables	179
11.6.4 Global Variables	179
11.6.5 Scope of Script and Global Variables	180
Variable Scope Scripts	181
11.6.6 User-Defined Variables	182
11.6.7 Variable Attributes	183
exists Attribute	183
length Attribute	183
11.6.8 Built-in Variables	184
_date	185
_echo	185
_execrc	185
_file	186
_halt	187
_keepalive	187
_lastmsg	187
_lastrc	188
_lines	189
_path	189
_rc	189
_time	190
_uuid	190
11.6.9 Logical Name Built-In Variables	191
Examples	191
11.7 if Statement	192
11.7.1 Comparison Operations	192
Comparators	192
EQ - Equal	193

NE - Not Equal	193
LT - Less Than	194
GT - Greater Than	194
LE - Less Than or Equal	194
GE - Greater Than or Equal	195
11.7.2 Adding an Alternate Path with else Statement	196
Alternate Path without else Statement	196
Alternate Path with else Statement	197
11.7.3 Nested Conditionals	198
11.7.4 Returning Early Using the return Command	199
11.8 while Statement	200
11.9 fordata Statement	201
Example	201
11.10 forfiles Statement	202
11.10.1 forfiles Built-In Variables	203
_file Variable Attributes	204
11.10.2 forfiles File Specification	205
11.10.3 Breaking Out Using the break Command	206
11.11 Creating In-Stream Data with the data Command	207
11.11.1 Creating an In-Stream Data Element	207
Example	208
11.11.2 Printing Data Element Information	208
Chapter 12 UDM Transfer Operations	209
12.1 Overview	209
12.2 Transfer Sessions	210
12.2.1 Opening a Transfer Session	210
Opening a Two-Party Transfer Session	210
Opening a Three-Party Transfer Session	211
12.2.2 Session Options	211
12.2.3 Closing a Session	212
12.3 File Systems	213
12.3.1 File System Overview	213
12.3.2 Changing the Current File System	214
12.4 UDM Common File System	215
12.4.1 Common File System Terminology	215
12.5 z/OS File System	217
12.6 OS/400 File Systems	219
12.6.1 HFS	219
12.6.2 LIB	220

12.7 Transfer Modes and Attributes	221
12.7.1 Setting the Transfer Type	221
12.7.2 Transfer Attributes	222
12.7.3 End of Line Sequence	223
eol Attribute	224
12.7.4 Line Length and Line Operations	225
12.8 Copying Files with UDM	226
12.8.1 Simple Copy Operation	226
Examples	226
12.8.2 Move Operation	227
12.8.3 Copying Multiple Files Using Wildcards	228
12.8.4 File Extension Attributes	229
12.8.5 File Creation Options	229
12.8.6 File Permission Attribute	230
Examples	230
Defaults	231
12.8.7 Destination umask	231
12.8.8 Transaction-Oriented Transfers	231
12.8.9 Changing the Current Directory in UDM	232
12.9 Auditing Transfer Operations	233
12.9.1 Logging File Transfer Operations	233
12.9.2 Reporting Transfer Progress	234
Chapter 13 Transfer Operations (z/OS-Specific)	235
13.1 Overview	235
13.2 z/OS I/O	236
13.2.1 Data Sets	236
Data Set Names	236
Data Set Organization	237
Record Format	237
Block Size	237
13.2.2 Generation Data Group and Generation Data Sets	238
Allocation	238
13.2.3 Catalogs	239
Symbolic Names	239
Catalog Entry Types	239
13.2.4 Allocation	239
13.3 UDM Commands under z/OS	240
13.3.1 attrib (Attribute) Command	240
13.3.2 cd (Change Directory) Command	244
DSN (data set name) File System	244
Examples	244

DD (ddname) File System	245
13.3.3 copy (Copy) Command	246
DSN File System	246
Sequential Data Sets	246
Partitioned Data Sets	247
DD File System	248
Sequential ddnames	248
Partitioned ddnames	249
13.4 Copying Load Modules	250
13.4.1 Example	251
13.4.2 Error Reporting	252
13.4.3 Special Attributes	252
Chapter 14 Transfer Operations (OS/400-Specific).....	253
14.1 Overview	253
14.2 OS/400 I/O	254
14.2.1 File Systems	254
14.2.2 HFS (for OS/400) File System	254
14.2.3 LIB File System	254
File Types	254
14.2.4 Data Physical Files Support	255
Caution about Text Mode Transfer of Files with DDS	255
14.2.5 Source Physical Files Support	256
14.2.6 Copying Source Physical Files	257
Like Copies of Source Physical File Data	257
Non-Source Physical to Source Physical Copies	257
Source Physical to Non-Source Physical Copies	257
14.2.7 Save Files Support	258
SAVF to SAVF Transfers	258
Non-SAVF to SAVF Transfers	258
SAVF to Non-SAVF Transfers	258
14.2.8 File Specifications	259
14.2.9 Wild Cards	259
Examples	259
14.3 Codepage - CCSID Mappings	260
14.3.1 CCSID Mapping	261
14.4 Command Reference	263
14.4.1 attrib (Attribute) Command	263
File Attributes	263
LIB File System Attributes	264
HFS Attributes	266
14.4.2 call (Call) Command	267

14.4.3 cd (Change Directory) Command	268
14.4.4 copy (Copy) Command	269
14.4.5 File Specification Rules	269
Source File Specification Rules	270
Destination File Specification Rules	271
14.4.6 delete (Delete) Command	276
delete Command Requirements	276
delete Command Forms	276
14.4.7 rename (Rename) Command	277
rename Command Requirements	277
rename Command Forms	278
Chapter 15 Remote Execution	279
15.1 Overview	279
15.2 exec Command	280
15.2.1 Executing Remote Commands within UDM	280
15.2.2 Return Values	282
15.2.3 exec Command Examples	282
15.3 execsap Command	283
15.3.1 Triggering SAP Events within UDM	283
15.3.2 execsap Command Example	284
Chapter 16 Return Code Processing	285
16.1 Overview	285
16.1.1 UDM Return Codes	285
16.2 Return Codes in UDM Built-In Variables	286
_lastrc Variable	286
_rc Variable	286
_haltom Variable	286
16.3 Setting Return Codes	287
16.3.1 Return Codes in set (Set) Command	287
Issuing the set Command	287
16.3.2 Return Codes in return (Return) Command	287
Appendix A Examples	288
A.1 Overview	288
A.2 UDM Manager for z/OS Examples	289
A.2.1 Copy a File to an Existing z/OS Sequential Data Set	290
DD file system	290
DSN file system	291
A.2.2 Copy a z/OS Sequential Data Set to a File	292

DD file system	292
DSN file system	292
A.2.3 Copy a Set of Files to an Existing z/OS Partitioned Data Set	293
DD file system	294
DSN file system	294
A.2.4 Copy a File to a New z/OS Sequential Data Set	295
DSN file system	295
A.2.5 Copy a Set of Files to a New z/OS Partitioned Data Set	296
DSN file system	296
A.3 UDM Manager for UNIX and Windows Examples	297
A.3.1 Simple File Copy to the Manager	298
A.3.2 Simple File Copy to the Server	299
A.3.3 Copy a Set of Files	300
A.4 UDM Manager for OS/400 Examples	301
A.4.1 Copy a File to an Existing OS/400 File	302
LIB file system	302
HFS file system	303
A.4.2 Copy an OS/400 Data Physical File to a File	304
LIB file system	304
A.4.3 Copy a Set of Files to an Existing Data Physical File	305
LIB file system	305
A.4.4 Copy a File to a New OS/400 Data Physical File	306
LIB file system	306
A.4.5 Copy a File to a New OS/400 Source Physical File	307
LIB file system	307
A.4.6 Copy a Set of Files to a New Data Physical File on OS/400	308
LIB file system	308
A.4.7 Copy Different Types of OS/400 Files using forfiles and \$(_file.type) ...	309
A.4.8 Invoke a Script from a Batch Job	310
LIB file system	310
Appendix B Customer Support	311

List of Figures

Chapter 1 Overview	34
Figure 1.1 UDM Transfer Sessions	37
Chapter 2 Features	38
Figure 2.1 Remote Configuration - Unmanaged and Managed Modes of Operation	50
Figure 2.2 Universal Configuration Manager Error dialog - Windows Vista	52
Figure 2.3 Windows Vista - Program Compatibility Assistant	53
Figure 2.4 Universal Configuration Manager	55
Figure 2.5 Universal Configuration Manager - UDM Manager	58
Figure 2.6 Universal Configuration Manager - UDM Server	59
Figure 2.7 X.500 Directory (sample)	89
Figure 2.8 X.509 Version 3 Certificate (sample)	90
Chapter 3 Universal Data Mover Manager for z/OS	94
Figure 3.1 UDM Manager for z/OS – JCL Procedure	96
Figure 3.2 UDM Manager for z/OS – JCL	98
Figure 3.3 UDM Manager for z/OS - Command Line Syntax (1 of 2)	103
Figure 3.4 UDM Manager for z/OS - Command Line Syntax (2 of 2)	104
Chapter 4 Universal Data Mover Manager for Windows	107
Figure 4.1 UDM Manager for Windows - Command Line Syntax	112
Chapter 5 Universal Data Mover Manager for UNIX	114
Figure 5.1 UDM Manager for UNIX - Command Line Syntax	119
Chapter 6 Universal Data Mover Manager for OS/400	122
Figure 6.1 UDM Manager for OS/400 - Command Line Syntax	129
Chapter 8 Universal Data Mover Server for Windows	140

Figure 8.1	Universal Configuration Manager - Component Definitions	141
Figure 8.2	Universal Configuration Manager - Universal Data Mover Server - Access ACL	147
Chapter 13	Transfer Operations (z/OS-Specific).....	235
Figure 13.1	Load Module Transfer Script - Example	251
Figure 13.2	Load Module Transfer Script - Output	251

List of Tables

Preface	30
Table P.1 Command Line Syntax	31
Chapter 2 Features	38
Table 2.1 UNIX Configuration File Directory Search	46
Table 2.2 Supported SSL cipher suites	62
Table 2.3 Component Communication States	72
Table 2.4 Certificate Map Matching Criteria	79
Table 2.5 Certificate Identifier Field	79
Table 2.6 Client IP Address - Matching Criteria	80
Table 2.7 Certificate Fields	91
Chapter 3 Universal Data Mover Manager for z/OS	94
Table 3.1 UDM Manager for z/OS – DD Statements in JCL	97
Table 3.2 UDM Manager for z/OS - Configuration Options	102
Chapter 4 Universal Data Mover Manager for Windows	107
Table 4.1 UDM Manager for Windows - Configuration Options	111
Chapter 5 Universal Data Mover Manager for UNIX	114
Table 5.1 UDM Manager for UNIX - Configuration Options	118
Chapter 6 Universal Data Mover Manager for OS/400	122
Table 6.1 UDM Manager for OS/400 - Configuration Options	128
Chapter 7 Universal Data Mover Server for z/OS	132
Table 7.1 UDM Server for z/OS - Component Definition Options	133
Table 7.2 UDM Server for z/OS - Configuration Options	136
Table 7.3 UDM Server for z/OS - UACL Entries	138

Chapter 8 Universal Data Mover Server for Windows	140
Table 8.1 UDM Server for Windows - Component Definition Options	142
Table 8.2 UDM Server for Windows - Configuration Options	144
Table 8.3 UDM Server for Windows - UACL Entries	146
Chapter 9 Universal Data Mover Server for UNIX	148
Table 9.1 UDM Server for UNIX - Component Definition Options	149
Table 9.2 UDM Server for UNIX - Configuration Options	151
Table 9.3 UDM Server for UNIX - UACL Entries	153
Chapter 10 Universal Data Mover Server for OS/400	155
Table 10.1 UDM Server for OS/400 - Component Definition Options	156
Table 10.2 UDM Server for OS/400 - Configuration Options	158
Table 10.3 UDMr Server for OS/400 - UACL Entries	160
Chapter 11 UDM Scripting Language	162
Table 11.1 UDM Commands	164
Table 11.2 UDM Command Expressions - Operators	169
Table 11.3 Built-In Variables	184
Table 11.4 _file Built-in Variable – Special Attributes	186
Chapter 12 UDM Transfer Operations	209
Table 12.1 CFS Terminology for Hierarchical File Systems	216
Table 12.2 CFS Terminology Associated with z/OS Data Sets	218
Table 12.3 CFS Terminology Associated with z/OS ddnames	218
Table 12.4 CFS Terminology Associated with LIB File Types	220
Chapter 13 Transfer Operations (z/OS-Specific)	235
Table 13.1 attrib Command - Dynamic Allocation Attributes	243
Table 13.2 cd Command in DSN File System	244
Table 13.3 copy Command File Specifications for Sequential Data Sets	246
Table 13.4 copy Command Destination File Specifications for Partitioned Data Sets	247
Table 13.5 copy Command Source File Specifications for Partitioned Data Sets	247
Table 13.6 copy Command Destination File Specifications for Sequential ddnames	248
Table 13.7 copy Command Source File Specification for Sequential ddnames	248
Table 13.8 copy Command Destination File Specifications for Partitioned ddnames	249
Table 13.9 copy Command Source File Specifications for Sequential ddnames	249
Chapter 14 Transfer Operations (OS/400-Specific)	253
Table 14.1 CCSID Mappings	262
Table 14.2 OS/400-Specific LIB File Attributes for Creating New Files	266
Table 14.3 OS/400 -Specific HFS File Attributes for Creating New Files	266
Table 14.4 delete Command Forms with UDM under OS/400	276

Table 14.5	rename Command Forms	278
Chapter 16 Return Code Processing		285
Table 16.1	UDM Return Codes	285

Preface

Document Structure

This document is written using specific conventions for text formatting and according to a specific document structure in order to make it as useful as possible for the largest audience. The following sections describe the document formatting conventions and organization.

Format

Starting with the Universal Products 3.2.0 release, the Universal Data Mover User Guide has been reformatted and restructured.

Most importantly, links to detailed information in a companion document, the Universal Data Mover Reference Guide, have been created in the user guide.

In order for the links between these documents to work correctly:

- Place the documents in the same folder.
- In Adobe Reader / Adobe Acrobat, de-select **Open cross-document link in same window** in the **General** category of your **Preferences** dialog (selected from the **Edit** menu).

Conventions

Specific text formatting conventions are used within this document to represent different information. The following conventions are used.

Typeface and Fonts

This Font identifies specific names of different types of information, such as file names or directories (for example, `\abc\123\help.txt`).

Command Line Syntax Diagrams

Command line syntax diagrams use the following conventions:

Convention	Description
bold monospace font	Specifies values to be typed verbatim, such as file / data set names.
<i>italic monospace font</i>	Specifies values to be supplied by the user.
[]	Encloses configuration options or values that are optional.
{ }	Encloses configuration options or values of which one must be chosen.
	Separates a list of possible choices.
...	Specifies that the previous item may be repeated one or more times.
BOLD UPPER CASE	Specifies a group of options or values that are defined elsewhere.

Table P.1 Command Line Syntax

Operating System-Specific Text

Most of this document describes the product in the context of all supported operating systems. At times, it is necessary to refer to operating system-specific information. This information is introduced with a special header, which is followed by the operating system-specific text in a different font size from the normal text.

z/OS

This text pertains specifically to the z/OS line of operating systems.

This text resumes the information pertaining to all operating systems.

Tips from the Stoneman



Stoneman's Tip

Look to the Stoneman for suggestions or for any other information that requires special attention.

Vendor References

References are made throughout this document to a variety of vendor operating systems. We attempt to use the most current product names when referencing vendor software.

The following names are used within this document:

- **z/OS** is synonymous with IBM z/OS and IBM OS/390 line of operating systems.
- **Windows** is synonymous with Microsoft's Windows 2000 / 2003 / 2008, Windows XP, Windows Vista, and Windows 7 lines of operating systems. Any differences between the different systems will be noted.
- **UNIX** is synonymous with operating systems based on AT&T and BSD origins and the Linux operating system.
- **OS/400** is synonymous with IBM OS/400, IBM i/5, and IBM i operating systems.
- **AS/400** is synonymous for IBM AS/400, IBM iSeries, and IBM System i systems.

Note: These names do not imply software support in any manner. For a detailed list of supported operating systems, see the Universal Products 3.2.0 Installation Guide.

Document Organization

- [Overview](#) (Chapter 1)
General architectural and functional overview of Universal Data Mover.
- [Features](#) (Chapter 2)
Description of Universal Data Mover features, including configuration methods and network protocols.
- [Universal Data Mover Manager for z/OS](#) (Chapter 3)
Description of Universal Data Mover Manager specific to the z/OS operating system.
- [Universal Data Mover Manager for Windows](#) (Chapter 4)
Description of Universal Data Mover Manager specific to the Windows operating system.
- [Universal Data Mover Manager for UNIX](#) (Chapter 5)
Description of Universal Data Mover Manager specific to the UNIX operating system.
- [Universal Data Mover Manager for OS/400](#) (Chapter 6)
Description of Universal Data Mover Manager specific to the OS/400 operating system.
- [Universal Data Mover Server for z/OS](#) (Chapter 7)
Description of Universal Data Mover Server specific to the z/OS operating system.
- [Universal Data Mover Server for Windows](#) (Chapter 8)
Description of Universal Data Mover Server specific to the Windows operating system.
- [Universal Data Mover Server for UNIX](#) (Chapter 9)
Description of Universal Data Mover Server specific to the UNIX operating system.
- [Universal Data Mover Server for OS/400](#) (Chapter 10)
Description of Universal Data Mover Server specific to the OS/400 operating system.
- [UDM Scripting Language](#) (Chapter 11)
Description of the Universal Data Mover scripting language.
- [UDM Transfer Operations](#) (Chapter 12)
General description of the Universal Data Mover transfer operations.
- [Transfer Operations \(z/OS-Specific\)](#) (Chapter 13)
Description of the Universal Data Mover transfer operations specific to the z/OS operating system.
- [Transfer Operations \(OS/400-Specific\)](#) (Chapter 14)
General description of the Universal Data Mover transfer operations specific to the OS/400 operating system.
- [Remote Execution](#) (Chapter 15)
Description of remote execution procedures for Universal Data Mover.
- [Return Code Processing](#) (Chapter 16)
Description of Universal Data Mover return code processing.
- [Examples](#) (Appendix A)
Examples of Universal Data Mover Manager for specific operating systems.
- [Customer Support](#) (Appendix B)
Customer support contact information for users of Universal Data Mover.

Chapter 1

Overview

1.1 Introduction to Universal Data Mover

This chapter provides general information on Universal Data Mover (UDM).

UDM is a secure and reliable data transfer solution developed specifically for corporate IT infrastructures and automated data center environments. UDM is a cost-effective alternative to the traditional complex and hard-to-implement offerings that makes transferring data between various enterprise and desktop platforms reliable and easy.

1.2 Transfer Components

There are three components to any UDM transfer operation:

1. Manager
2. Primary server
3. Secondary server

The manager may act as the primary server, depending on the type of transfer session: two-party or three-party (see Section [1.3 Transfer Sessions](#)). The secondary server is always a separate and distinct component invoked via the Universal Broker.

1.2.1 Manager

The UDM Manager processes commands using UDM's scripting language. The UDM Manager receives commands from the user through an interactive session, an external script file, or some combination of the two. Before the UDM Manager can initiate any transfer operations, it must first establish a transfer session where it invokes the primary and secondary servers, which actually conduct the transfer operations.

1.2.2 Primary Server

When a transfer session is being established, the UDM Manager invokes the primary server, which acts as the first endpoint in a transfer operation. In turn, the primary server invokes the secondary server, providing a single path of communication. The primary server also acts a relay for the UDM Manager, forwarding on any messages for the secondary server from the UDM Manager. This single message pipeline reduces the number of connections needed for three-party transfers (see Section [1.3.3 Three-Party Transfer Sessions](#)).

1.2.3 Secondary Server

The secondary server acts as the second endpoint in a transfer operation. Data is transferred between primary and secondary servers, with either endpoint able to act as the source in a transfer operation.

1.3 Transfer Sessions

As discussed in Section [1.2 Transfer Components](#), transfer operations take place within the context of a transfer session. A transfer operation is initiated once the UDM Manager has established a transfer session with the primary and secondary transfer servers. All subsequent transfer operations take place between the primary and secondary transfer servers.

UDM transfer sessions can be either two-party or three-party.

1.3.1 Logical Names

When a transfer session is established, the user gives each server a unique logical name. Commands addressed to a particular server reference this logical name.

1.3.2 Two-Party Transfer Sessions

For a two-party transfer session, the UDM Manager also acts as the primary transfer server, running in the directory – and under the user ID – under which the UDM Manager was launched. This means that the machine on which UDM Manager resides is the first endpoint of the transfer.

With a two-party transfer session, the secondary server is invoked by the manager / primary server via the Universal Broker. The second endpoint of the transfer session will be on the machine in which the secondary server was spawned. Transfer operations occur between the manager / primary server and the secondary server.

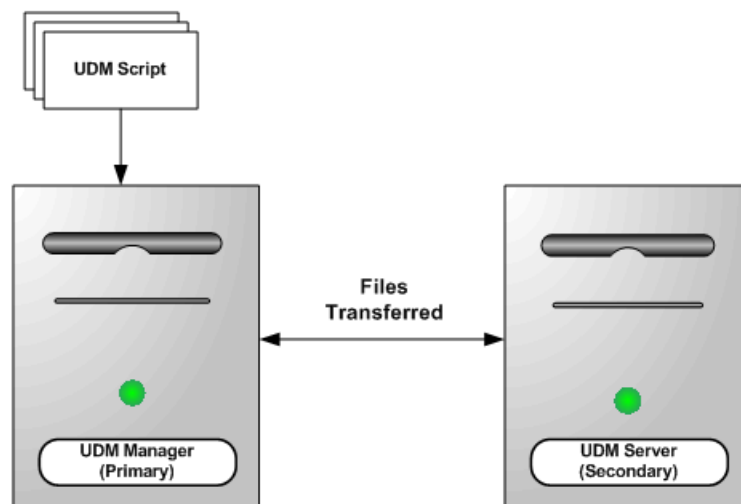
(See [Figure 1.1 UDM Transfer Sessions](#).)

1.3.3 Three-Party Transfer Sessions

For a three-party transfer session, the UDM Manager acts solely as a control point for transfer operations, sending commands to the primary and secondary servers to be executed. Both the primary and secondary servers are spawned via the Universal Broker, and transfer operations take place between the two machines under which these servers are running.

(See [Figure 1.1 UDM Transfer Sessions](#).)

Two-Party Transfer



Three-Party Transfer

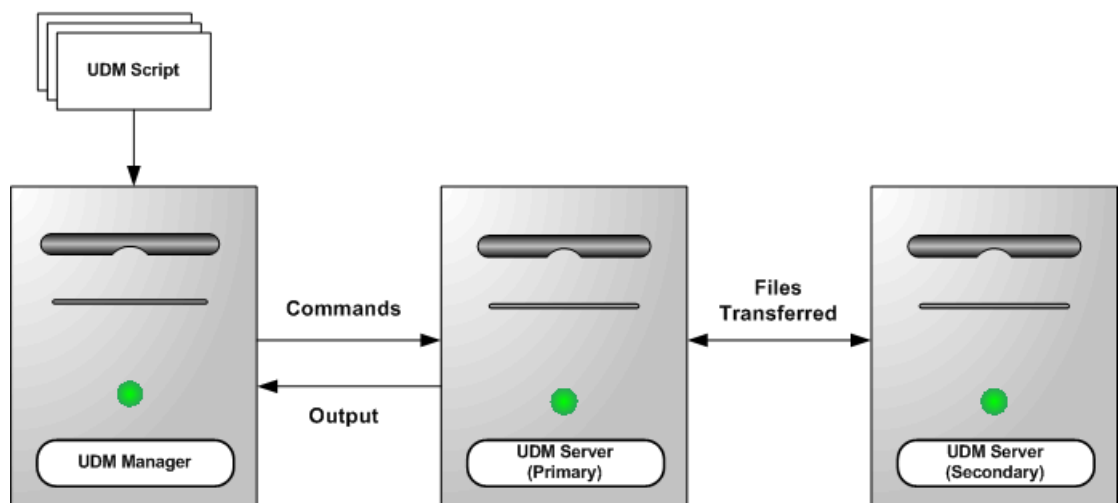


Figure 1.1 UDM Transfer Sessions

Chapter 2

Features

2.1 Overview

This chapter provides information on Universal Data Mover (UDM) features that apply to all operating systems.

- [Configuration](#)
- [Universal Configuration Manager](#)
- [Remote Configuration](#)
- [Network Data Transmission](#)
- [Fault Tolerance](#)
- [z/OS CANCEL Command Support](#)
- [Universal Access Control List](#)
- [Message and Audit Facilities](#)
- [X.509 Certificates](#)

2.2 Configuration

Product configuration consists of specifying options that control product behavior and resource allocation.

- An example of configurable product behavior is whether or not data transferred over the network is compressed.
- An example of configurable resource allocation is the directory location in which the product creates its log files.

Each option is comprised of a pre-defined parameter, which identifies the option, and one or more values. The format of the parameter depends on the method being used to specify the option.

Although there are many configurable product options, Universal Products, in general, are designed to require minimal configuration and administration. The default options will work very well in most environments. When local requirements do require a change in product configuration, there are multiple methods available to configure the products in order to meet your needs.

2.2.1 Configuration Methods

All Stonebranch Inc. Universal Products provide a consistent and flexible method of configuration. An operating system's native configuration methods, such as configuration files, are utilized in order to integrate with existing system management policies and procedures for the platform.

Depending on specific Universal Products, and the operating system on which it is being run, product configuration is performed by one or more methods. These configuration methods, in their order of precedence, are:

1. [Command Line](#)
2. [Command Line File](#)
3. [Environment Variables](#)
4. [Configuration File](#)

This order of precedence means that a command option specified on the command line overrides the same option specified in a command file, which overrides the same option specified with an environment variable, which overrides the same option specified with a configuration file keyword.

Note: For security reasons, not all options can be overridden.

2.2.2 Command Line

Command line options affect one instance of a program execution. Each time that you execute a program, command line options let you tailor the behavior of the program to meet the specific needs for that execution.

Command line options are the highest in order of precedence of all the configuration methods (see Section [2.2.1 Configuration Methods](#)). They override the options specified using all other configuration methods, except where indicated.

Command line options consist of:

- Parameter (name of the option)
- Value (pre-defined or user-defined value of the option)

The command line syntax depends, in part, on the operating system, as noted below.

An value may or may not be case-sensitive, depending on what it is specifying. For example, if a value is either *yes* or *no*, it is not case-sensitive. It could be specified as *YES*, *Yes*, or *yes*. However, if a value specifies a directory name or file name, it would be case-sensitive if the operating system's file system is case-sensitive.

If an option is specified more than once on the command line, the last instance of the option specified is used.

z/OS

z/OS command line options are specified in the JCL EXEC statement PARM keyword or on the SYSIN ddname. The PARM keyword is used to pass command line options to the program being executed with the EXEC statement.

Command line options are prefixed with a dash (-) character. For many options, there are two different forms in which they can be specified:

- Short form: one case-sensitive character
- Long form: two or more case-insensitive characters

The parameter and value must be separated by at least one space.

Example command line options specified in the PARM value follow:

Short form:

```
PARM='-I INFO -G yes'
```

Long form:

```
PARM='-LEVEL INFO -LOGIN YES'
```

As noted above, z/OS command line options also can be specified on the SYSIN ddname. This is the easiest and least restrictive place to specify options, since the PARM values are limited in length. The options specified in the SYSIN ddname have the same syntax. Options can be specified on one line or multiple lines. The data set or inline data allocated to the SYSIN ddname cannot have line numbers in the last 8 columns (that is, all columns of the records are used as input).

UNIX and Windows

UNIX and Windows command line options are prefixed with a dash (-) character, and alternatively on Windows, the slash (/) character.

For many options, there are two different forms in which they can be specified:

- Short form: one case-sensitive character.
- Long form: two or more case insensitive characters.

The parameter and value must be separated by at least one space or tab character.

Example command line options follow:

Short form:

```
-l info -G yes
```

Long form:

```
-level info -login yes
```

```
-LEVEL info -LoGiN YES
```

OS/400

OS/400 command line options use the native conventions for Command Language (CL) commands. The option name is specified as a CL parameter with its value enclosed in parentheses.

Example command line options follow:

Command line options:

```
MSGLEVEL(*info) COMPRESS(*yes)
```

All of the Stonebranch Inc. Universal Products provide OS/400-style command panels. The panels are accessed by entering the command name on the command line and pressing the F4 (PROMPT) key.

2.2.3 Command Line File

The command line file contains command line options specified in a file. The command line file enables you to save common command line options in permanent storage and reference them as needed.

The command line file is the second to highest in the precedence order after command line options (see Section [2.2.1 Configuration Methods](#)).

Individual command line options can be specified on one or multiple lines. Blank lines are ignored. Lines starting with the hash (#) character are ignored and can be used for comments.

The command line file can be encrypted if it is necessary to secure the contents.

Note: If the contents of the file contain sensitive material, the operating system's native file and user security facilities should be used in addition to the file encryption provided by the Universal Products.

2.2.4 Environment Variables

Environment variables, like command line options, allow options to be specified for one instance of a program execution. Each time that you execute a program, environment variables allow you to tailor the behavior of the program to meet the specific needs for that execution.

Environment variables are the third to highest in the precedence order after command line file options (see [Section 2.2.1 Configuration Methods](#)).

Each operating system has its own unique method of setting environment variables.

All environment variables used by Universal Products are upper case and are prefixed with a product identifier consisting of three or four characters. The product sections specify the value of the environment variables. Values are case-sensitive.

z/OS

Environment variables are specified in the JCL EXEC statement PARM keyword. Environment variables are part of the IBM Language Environment (LE) and as such are specified as LE runtime options. The PARM value is divided into LE options and application options by a slash (/) character. Options to the left of the slash are LE options and options to the right are application options.

Example of setting an environment variable:

```
Set option UDMLEVEL to a value of INFO:  
PARM=' ENVAR("UDMLEVEL=INFO")/'
```

UNIX

Environment variables in UNIX are defined as part of the shell environment. As such, shell commands are used to set environment variables. The environment variable must be exported to be used by a called program.

Example of setting an environment variable:

```
Set option UDMLEVEL to a value of INFO in a bourne, bash, or korn shell:  
UDMLEVEL=INFO  
export UDMLEVEL
```

Windows

Environment variables in Windows are defined as part of the Windows console command environment. As such, console commands are used to set environment variables.

Example of setting an environment variable:

Set option UDMLEVEL to a value of INFO:

```
SET UDMLEVEL=INFO
```

OS/400

Environment variables in OS/400 are defined with Command Language (CL) commands for the current job environment.

Example of setting an environment variable:

Set option UDMLEVEL to a value of INFO:

```
ADDENVVAR ENVVAR(UDMLEVEL) VALUE(INFO)
```

2.2.5 Configuration File

Configuration files are used to specify system-wide configuration values. They are last in precedence order for specifying configuration options (see Section [2.2.1 Configuration Methods](#)).

(For most Universal Products, some options can be specified only in a configuration file, while other options can be overridden by individual command executions. The Stonebranch, Inc. documentation for each product identifies these options.)

If an option is specified more than once in a configuration file, the last option specified is used.

All configuration files on a system are maintained by the local Universal Broker. The Universal Broker serves the configuration data to other Universal Products running on the local system. The one exception is Universal Enterprise Controller (UEC). UEC directly reads its own configuration files.

The Universal Broker reads the configuration files when it first starts or when it receives a REFRESH command from Universal Control or Universal Enterprise Controller. Any changes made to a configuration file are not in effect until the Broker is recycled or receives a REFRESH command.

Universal Product components do not read the configuration files themselves. When a component starts, it first registers with the locally running Universal Broker. As part of the registration process, the Broker returns the components configuration data.

When the Universal Broker is operating in managed mode, the configuration information for the various Universal Products is "locked down" and can be modified or viewed only via the Universal Management Console (see Section [2.3.2 Managed Mode](#)).

z/OS

Configuration files are members of a PDSE. The data set record format is fixed or fixed block with a record length of 80. No line numbers can exist in columns 72-80. All 80 columns are processed as data.

All configuration files are installed in the **UNVCONF** library.

See Section [2.2.6 Configuration File Syntax](#) for the configuration file syntax.

UNIX

Configuration files are regular text files on UNIX.

Universal Broker searches for the configuration files in a fixed list of directories. The Broker will use the first configuration file that it finds in its search. The directories are listed below in the order they are searched:

Directory	Notes
/etc/opt/universal	
/etc/universal	Installation default
/etc/stonebranch	Obsolete as of version 2.2.0
/etc	
/usr/etc/universal	
/usr/etc/stonebranch	Obsolete as of version 2.2.0
/usr/etc	

Table 2.1 UNIX Configuration File Directory Search

See [2.2.6 Configuration File Syntax](#) for the configuration file syntax.

Windows

Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options.

The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values (see [Section 2.4 Universal Configuration Manager](#)).

OS/400

The configuration files on OS/400 are stored in a source physical file named UNVCONF in the UNVPRD320 library. The files can be edited with a text editor.

See [Section 2.2.6 Configuration File Syntax](#) for the configuration file syntax.

2.2.6 Configuration File Syntax

Configuration files are text files that can be edited with any available text editor.

The following rules apply for configuration file syntax:

- Options are specified in a keyword / value format.
 - Keywords can start in any column.
 - Keywords must be separated from values by at least one space or tab character.
 - Keywords are not case sensitive.
 - Keywords cannot contain spaces or tabs.
 - Values can contain spaces and tabs, but if they do, they must be enclosed in single (') or double (") quotation marks. Repeat the enclosing characters to include them as part of the value.
 - Values case sensitivity depends on the value being specified. For example:
 - Directory and file names are case sensitive.
 - Pre-defined values (such as **yes** and **no**) are not case sensitive.
 - Each keyword / value pair must be on one line.
 - Characters after the value are ignored.
 - Newline characters are not permitted in a value.
 - Values can be continued from one line to the next either by ending the line with a:
 - Plus (+) character, to remove all intervening spaces.
 - Minus (-) character, to preserve all intervening spaces between the end of the line being continued and the beginning of the continuing line.
- Ensure that the line continuation character is the last character on a line.
- Comment lines start with a hash (#) character.
 - Blank lines are ignored.

Note: If an option is specified more than once in a configuration file, the last option specified is used.

2.3 Remote Configuration

Universal Products can be configured remotely by Universal Enterprise Controller using the Universal Management Console (UMC) client application. UMC instructs the Universal Broker of a remote Universal Agent to modify the configurations of the Universal Products components managed by that Broker.

Universal Broker supports remote configuration in either of two modes:

1. [Unmanaged Mode](#)
2. [Managed Mode](#)

2.3.1 Unmanaged Mode

Unmanaged mode is the default mode of operations for Universal Broker. It allows a Universal Broker – and the Universal Products managed by that Universal Broker – to be configured either:

- Locally, by editing configuration files.
- Remotely, via UMC.

The system administrator for the machine on which a Universal Agent resides can use any text editor to modify the configuration files of the various local Universal Products.

Via UMC, selected users can modify all configurations of any Universal Agent, including the local Universal Agent. UMC sends the modified data to the Universal Broker of that agent, which Universal Broker then uses to update the appropriate configuration files.

If UMC sends modifications for a Universal Broker configuration, Universal Broker validates the modified data before it accepts it. If the data fails validation, Universal Broker does not update its configuration file.

If UMC sends modification to the configuration of any other Universal Products component, the Universal Broker updates the appropriate configuration file. The component will use this new configuration at its next invocation.

Note: If errors or invalid configuration values are updated via UMC for a component other than Universal Broker, the component may not run successfully until the configuration has been corrected.

2.3.2 Managed Mode

When a Universal Broker is operating in managed mode, the configuration information for all Universal Products components managed by that Universal Broker is "locked down." Universal Broker stores the information in a database file located within its specified spool directory. The information can be modified only via Universal Management Console (UMC).

From this point on, Universal Broker uses the database file – not the configuration files – to access configuration information. Any configuration changes made to the components – via UMC – are placed in the database file. Therefore, as long as Universal Broker stays in managed mode, the configuration files may no longer contain current or valid configuration information.

If managed mode is de-selected for the Universal Broker, it reads the database file where it stored the configuration information. Universal Broker uses this information to create and/or update configuration files for the components.

- If a configuration file exists in the configuration directory, it is overwritten.
- If a configuration file does not exist, it is created.

Note: Because of remote configuration and the desire to be able to "lock down" all product configurations, Universal Broker – and all Universal Products servers – no longer support the command line and environmental variables methods of specifying configuration options.

Selecting Managed Mode

The managed mode of operations for Universal Broker is selected via the Universal Enterprise Controller Administration client application.

(See the Universal Enterprise Controller 3.2.0 Client Applications guide for specific information on how to select managed mode.)

Figure 2.1, below, illustrates remote configuration for one Universal Agent in managed mode and one Universal Agent in unmanaged mode.

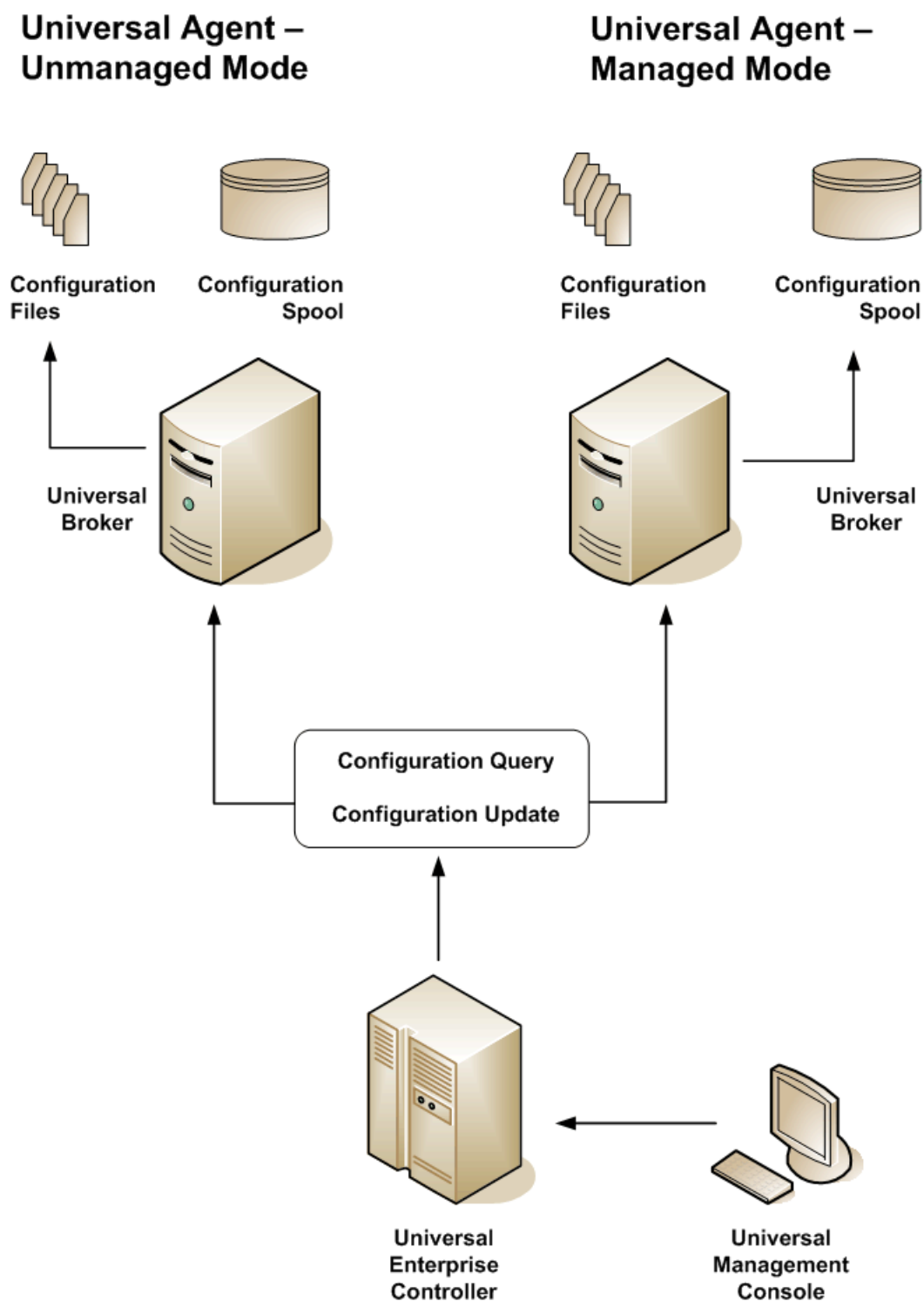


Figure 2.1 Remote Configuration - Unmanaged and Managed Modes of Operation

2.3.3 Universal Broker Startup

At Universal Broker start-up, in both managed and unmanaged modes, the Universal Broker configuration file always is read.

Unmanaged Mode

At Universal Broker start-up in unmanaged mode, Universal Broker reads the configuration files of all Universal Products components into its memory. The Universal Broker configuration file is used to define the Universal Broker configuration, just as all configuration files are used in unmanaged mode. Universal Broker updates its memory from the configuration files whenever Universal Control issues a REFRESH request.

Managed Mode

At Universal Broker start-up in managed mode, the Universal Broker configuration file points Universal Broker to the location of the configuration spool file, from which the Broker retrieves configuration information for all Universal Products. Universal Broker updates its memory from the configuration spool file and, automatically, after changes are made via UMC.

If more configuration information than needed is included in the Universal Broker configuration file at Universal Broker start-up, Universal Broker will update its running configuration with the information that it retrieved from the spool file. The configuration file that was used at start-up is made obsolete.

2.4 Universal Configuration Manager

The Universal Configuration Manager is a Universal Products graphical user interface application that enables you to configure all of the Universal Products that have been installed on a Windows operating system.

It is the recommended method of specifying configuration data that will not change with each command invocation. Universal Configuration Manager helps protect the integrity of the configuration file by validating all changes to configuration option values.

The configuration data for a Universal Products for Windows system is stored in the configuration file.

2.4.1 Availability

Universal Configuration Manager is installed automatically on the Windows operating system as part of every Universal Products for Windows installation.

It is available to all user accounts in the Windows Administrator group.

Windows Vista

When opening the Universal Configuration Manager for the first time on Windows Vista, two new operating system features, the Program Compatibility Assistant (PCA) and User Account Control (UAC), may affect its behavior.

With these two features enabled, the expected Universal Configuration Manager behavior is as follows:

1. Universal Configuration Manager may issue the following error:

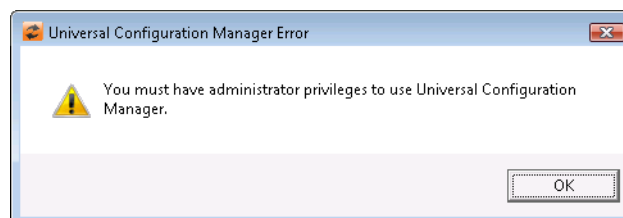


Figure 2.2 Universal Configuration Manager Error dialog - Windows Vista

2. Click **OK** to dismiss the error message.

The Windows Vista Program Compatibility Assistant (PCA) displays the following dialog:

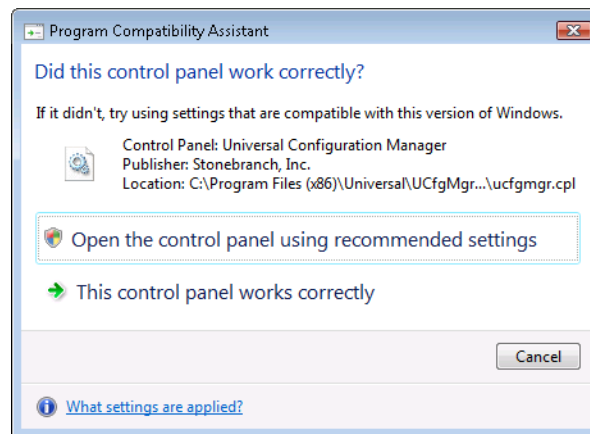


Figure 2.3 Windows Vista - Program Compatibility Assistant

3. To continue, select **Open the control panel using recommended settings**. This instructs the PCA to "shim" (Microsoft term) the Configuration Manager, establishing it as an application that requires elevated privileges.
Windows Vista User Account Control (UAC) then displays a prompt seeking permission to elevate the logged-in account's access token.
4. Select **Continue** to give the account full administrative privileges.
Subsequent attempts to open Universal Configuration Manager should result only in the UAC prompt.

2.4.2 Accessing the Universal Configuration Manager

To access the Universal Configuration Manager:

1. Click the **Start** icon at the lower left corner of your Windows operating system screen to display the Start menu.
2. Click (Settings/) **Control Panel** on the Start menu to display the Control Panel screen.
3. Select the Universal Configuration Manager icon to display the Universal Configuration Manager screen (see [Figure 2.4](#)).

Windows XP, Windows Vista, Windows Server 2008

Newer versions of Windows support a Control Panel view that places applet icons within categories. This "category view" may affect the location of the Universal Configuration Manager icon.

For example, the Windows XP Category View places the Universal Configuration Manager icon under the **Other Control Panel Options** link. Windows Vista and Windows Server 2008 place the icon within the **Additional Options** category.

If you have trouble locating the Universal Configuration Manager icon, simply switch to the Classic View to display all Control Panel icons at the same time.

64-bit Windows Editions

The Windows Control Panel places icons for all 32-bit applets under the **View x86 Control Panel Icons** (or, on newer versions, the **View 32-bit Control Panel Icons**) category, even when the Classic View is enabled.

When using the Category View, look for the 32-bit Control Panel applet icons in the **Additional Options** category.

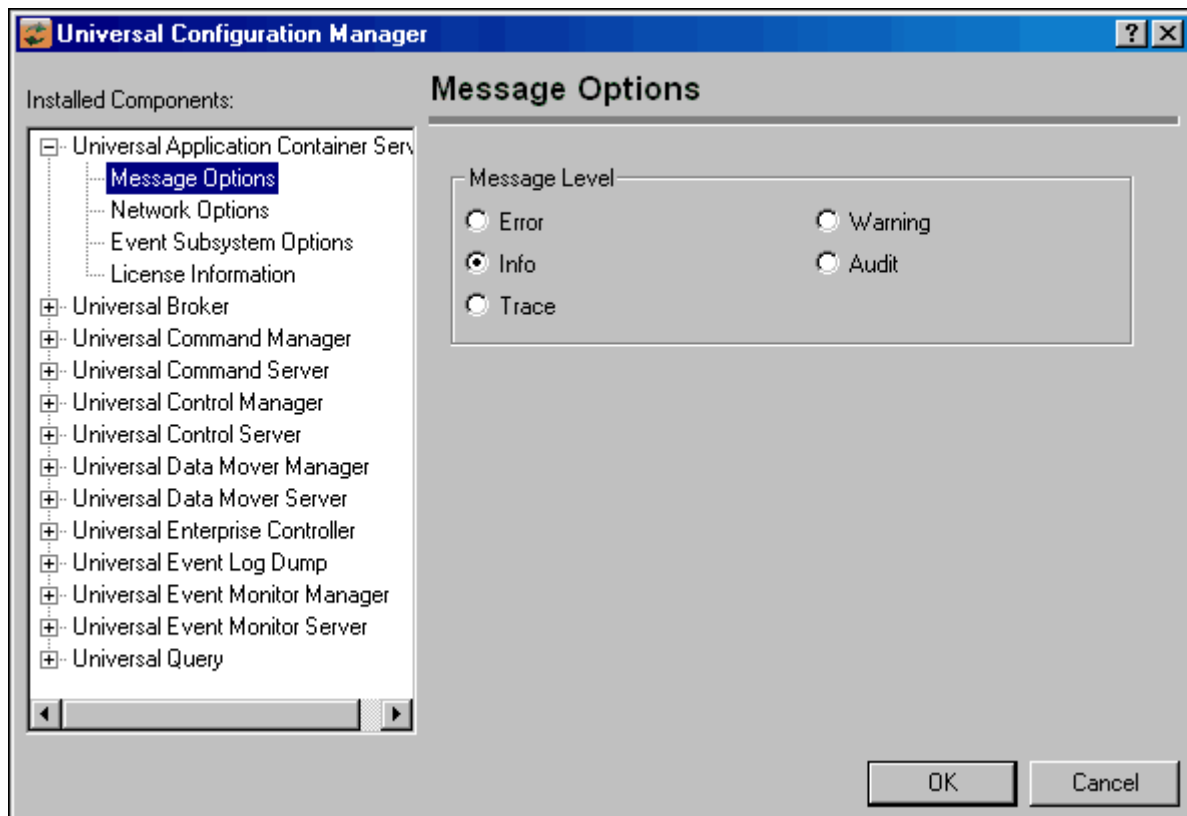


Figure 2.4 Universal Configuration Manager

Each Universal Configuration Manager screen contains two sections:

1. Left side of the screen displays the Installed Components tree, which lists:
 - Universal Products components currently installed on your system.
 - Property pages available for each component (as selected), which include one or more of the following:
 - Configuration options
 - Access control lists
 - Licensing information
 - Other component-specific information
2. Right side of the screen displays information for the selected component / page.

(By default, Universal Configuration Manager displays the first property page of the first component in the Installed Components tree.)

2.4.3 Navigating through Universal Configuration Manager

To display general information about a component, click the component name in the Installed Components list.

To display the list of property pages for a component, click the + icon next to the component name in the Installed Components list.

To display a property page, click the name of that page in the Installed Components list.

If a property page has one or more of its own pages, a + icon displays next to the name of that property page in the Installed Components list. Click that + icon to display a list of those pages.

In [Figure 2.4](#), for example:

- List of property pages is displayed for Universal Broker.
- Message Options property page has been selected, and information for that property is displayed on the right side of the page.
- No + icons next to any of the property pages indicates that they do not have one or more of their own property pages.

2.4.4 Modifying / Entering Data

On the property pages, modify / enter data by clicking radio buttons, selecting from drop-down lists, and/or typing in data entry fields.

Some property pages provide panels that you must click in order to:

- Modify or adjust the displayed information.
- Display additional, modifiable information.

Note: You do not have to click the **OK** button after every modification or entry, or on every property page on which you have modified and/or entered data. Clicking **OK** just once, on any page, will save the modifications and entries made on all pages – and will exit Universal Configuration Manager (see [Section 2.4.5 Saving Data](#).)

Rules for Modifying / Entering Data

The following rules apply for the modification and entry of data:

- Quotation marks are not required for configuration values that contain spaces.
- Edit controls (used to input free-form text values) handle conversion of any case sensitive configuration values. Except where specifically noted, values entered in all other edit controls are case insensitive.

2.4.5 Saving Data

To save all of the modifications / entries made on all of the property pages, click the **OK** button at the bottom of any property page. The information is saved in the configuration file, and Universal Broker is automatically refreshed.

Clicking the **OK** button also exits the Universal Configuration Manager. (If you click **OK** after every modification, you will have to re-access Universal Configuration Manager.)

To exit Universal Configuration Manager without saving any of the modifications / entries made on all property pages, click the **Cancel** button.

2.4.6 Accessing Help Information

Universal Configuration Manager provides context-sensitive help information for the fields and panels on every Universal Products component options screen.

To access Help:

1. Click the question mark (?) icon at the top right of the screen.
2. Move the cursor (now accompanied by the ?) to the field or panel for which you want help.
3. Click the field or panel to display Help text.
4. To remove the displayed Help text, click anywhere on the screen.

Windows Vista, Windows Server 2008

The Universal Configuration Manager's context-sensitive help is a WinHelp file, which Windows Vista and Windows Server 2008 does not support.

Microsoft offers the 32-bit WinHelp engine as a separate download from its website. If you require access to the Universal Configuration Manager's context-sensitive help, simply download and install the WinHelp engine.

2.4.7 Universal Data Mover Installed Components

Universal Data Mover Manager

Figure 2.5 illustrates the Universal Configuration Manager screen for the Universal Data Mover Manager.

The Installed Components list identifies all of the UDM Manager property pages.

The text describes the selected component, Universal Data Mover Manager.

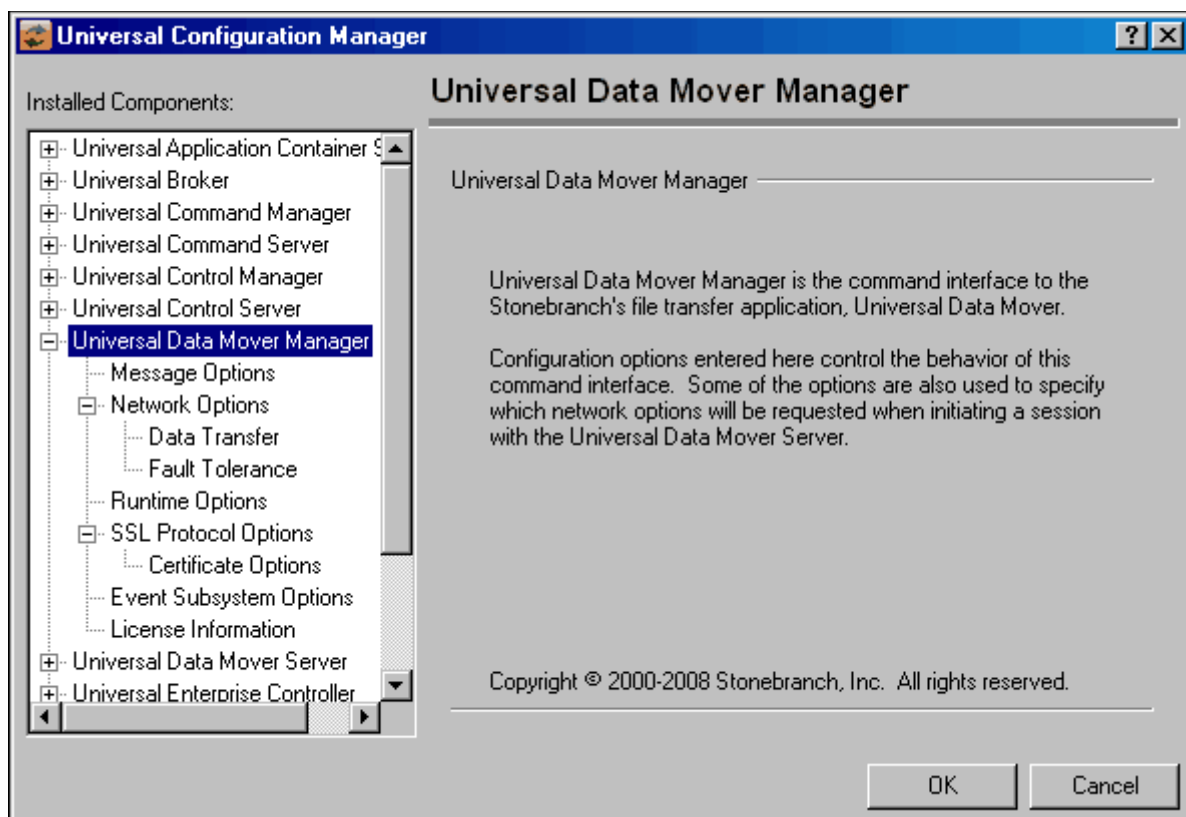


Figure 2.5 Universal Configuration Manager - UDM Manager

Universal Data Mover Server

Figure 2.6 illustrates the Universal Configuration Manager screen for the Universal Data Mover Server.

The Installed Components list identifies all of the UDM Server property pages.

The text describes the selected component, Universal Data Mover Server.

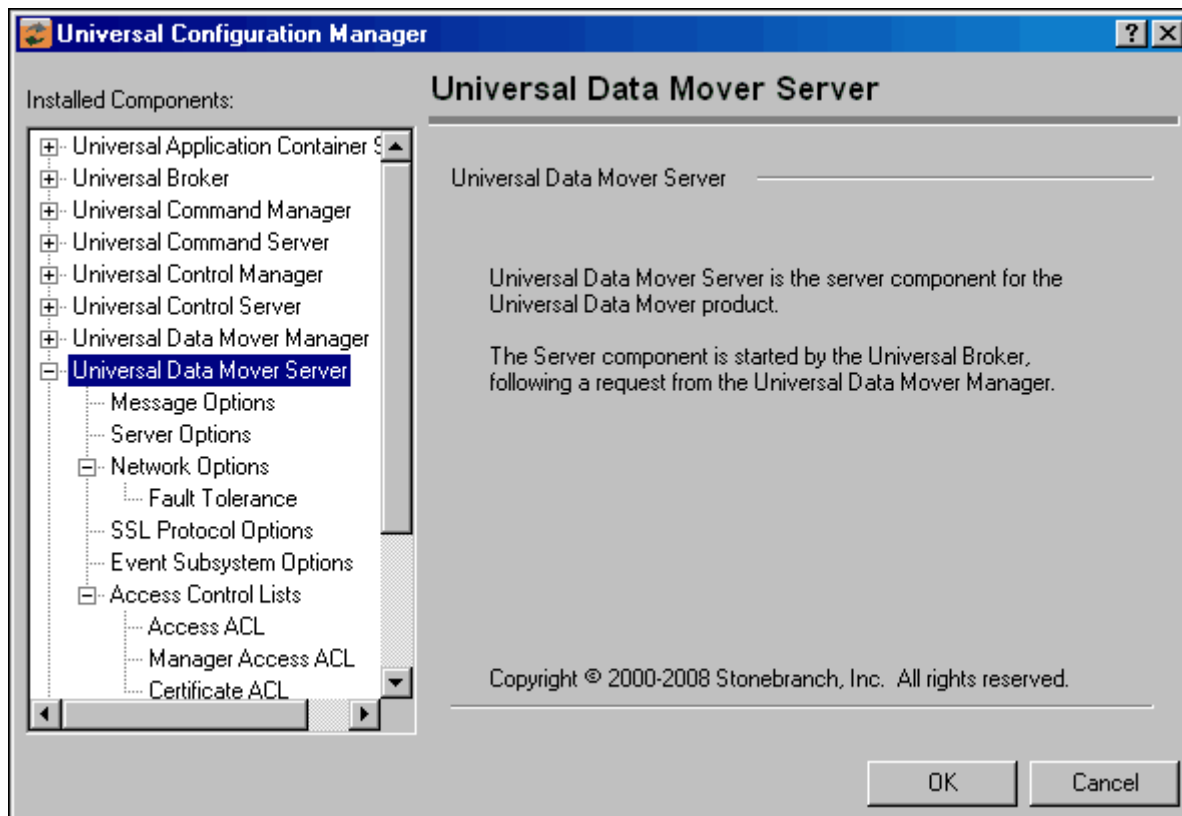


Figure 2.6 Universal Configuration Manager - UDM Server

2.5 Network Data Transmission

Distributed systems, such as Universal Data Mover, communicate over data networks. All Stonebranch products communicate using the TCP/IP protocol. The UDP protocol is not used for any product data communication over a network.

The Universal Products suite can utilize one of two network protocols:

1. Secure Socket Layer version 3 (SSLv3) provides the highest level of security available. SSL is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks.
2. Universal Products version 2 (UNVv2) legacy protocol is provided for backward compatibility with previous versions of Universal Products.

The following sections discuss each of the protocols.

In addition to the network protocol used to transmit data, Universal Products application protocol is discussed as well.

2.5.1 Secure Socket Layer Protocol

Universal Products implement the SSL protocol using either:

- OpenSSL library.
- IBM z/OS System SSL library, available on the z/OS operating system.

The most recent SSL standard is version3. A subsequent version was produced changing the name to Transport Layer Security version 1 (TLSv1).

TLSv1 is the actual protocol used by Universal Products. TLSv1 is more commonly referred to simply as SSL and the term SSL is used throughout the rest of this documentation to mean TLSv1 unless otherwise noted.

The SSL protocol addresses the major challenges of communicating securely over a potentially insecure data network. The following sections discuss the issue of data privacy and integrity, and peer authentication.

Data Privacy and Integrity

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. What they do with the data is up to them.

Data sent over the network that should remain private must be encrypted in a manner that unauthorized persons cannot determine what the original data contained regardless of their level of expertise, access to network resources, amount of data captured, and amount of time they have. The only party that should be able to read the data is the intended recipient.

As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data while it is transmitted over the network. The changed data may or may not be detected by the recipient depending on what changed and how it is processed. It may be accepted as valid data, but the information it represents is now erroneous.

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insures that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

Encryption algorithms are used to encrypt data into an unreadable format. The encryption process is computationally expensive. There are a variety of encryption algorithms some of which perform better than others. Some algorithms offer a higher level of security than others. Typically, the higher level of security requires more computational resources.

Message digest algorithms are used to produce a Message Authentication Code (MAC) that uniquely identifies a block of data. The sender computes a MAC for the data being sent based on a shared secret key the sender and receiver hold. The sender sends the data and the MAC to the receiver. The receiver computes a new MAC for the received data based on the shared secret key. If the two MAC's are the same, data integrity is maintained, else the data is rejected as it has been modified. Message digest algorithms are often referred to as MAC's and can be used synonymously in most contexts.

The SSL standard defines a set of encryption and message digest algorithms referred to cipher suites that insure data privacy and data integrity. Cipher suites pair encryption algorithms with appropriate message digest algorithms. The two algorithms cannot be specified individually.

Universal Products supports a subset of the complete SSL cipher suites defined by the standard. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

Table 2.2, below, identifies the supported cipher suites.

Cipher Suite Name	Description
RC4-SHA	128-bit RC4 encryption with SHA-1 message digest
RC4-MD5	128-bit RC4 encryption with MD5 message digest
AES256-SHA	256-bit AES encryption with SHA-1 message digest
AES128-SHA	128-bit AES encryption with SHA-1 message digest
DES-CBC3-SHA	128-bit Triple-DES encryption with SHA-1 message digest
DES-CBC-SHA	128-bit DES encryption with SHA-1 message digest
NULL-SHA	No encryption with SHA-1 message digest
NULL-MD5	No encryption with MD5 message digest

Table 2.2 Supported SSL cipher suites

Universal Products support one additional cipher suite name that is not part of the SSL protocol. The NULL-NULL cipher suite turns SSL off completely and instead uses the Universal Products Protocol (UNVv2) (see Section 2.5.2 Universal Products Protocol).

To turn off SSL, specify NULL-NULL for the data cipher list for all UDM servers used for the session and for the encrypt parameter on the [open](#) command.

Peer Authentication

When communicating with a party across a data network, how do you insure that the party you are communicating with (your peer) is who you believe? A common form of network attack is a malicious user representing themselves as another user or host.

Peer authentication insures that the peer is truly who they identify themselves as. Peer authentication applies to users, computer programs and hardware systems.

SSL uses X.509 certificates and public and private keys to identify an entity. An entity may be a person, a program, or a system. A complete description of X.509 certificates is beyond the scope of this documentation. Section 2.10 X.509 Certificates provides an overview to help get the reader oriented to the concepts, terminology and benefits.

For additional details, the following web site is recommended:

<http://www.faqs.org/rfcs/rfc3280.html>

2.5.2 Universal Products Protocol

The Universal Products protocol (UNVv2) is a proprietary protocol that securely and efficiently transports data across data networks. UNVv2 is used in Universal Products prior to version 3 and will be available in future versions.

UNVv2 addresses data privacy and integrity. It does not address peer authentication.

Data Privacy and Integrity

Data privacy is insured with data encryption algorithms. UNVv2 utilizes 128-bit RC4 encryption for all data encryption.

Data integrity is insured with message digest algorithms. UNVv2 utilizes 128-bit MD5 MAC's for data integrity. UNVv2 referred to data integrity as data authentication.

Encryption and integrity may be enabled and disabled on an individual bases.

Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each and every network session.

2.5.3 Universal Products Application Protocol

Universal Product components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- [Low-Overhead](#)
- [Secure](#)
- [Extensible](#)
- [Configurable Attributes](#)

The following sections refer to two categories of data transmitted by Universal Products:

- Control data (or messages) consists of messages generated by Universal Products components in order to communicate with each other. The user of the product has no access to the control data itself.
- Application data (or messages) consists of data that is transmitted as part of the requested work being executed. For example, standard input and output data of jobs Universal Command executes. The data is created by the job and read or written by Universal Command on behalf of the job.

Low-Overhead

The protocol is lightweight, in order to minimize its use of network bandwidth. The product provides application data compression options, which reduces the amount of network data even further.

There are two possible compression methods:

- **ZLIB** method offers the highest compression ratios with highest CPU utilization.
- **HASP** method offers the lowest compression ratios with lowest CPU utilization.

Note: Control data is not compressed. Compression options are available for application data only.

Secure

The protocol is secure. All control data exchanged between Universal Products components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: SSL and UNVv2.

The security features used in the control messages are not optional. They cannot be turned off. The security features are optional for application data sent over the network.

The data encryption options affect the application data being sent over the network. Special fields, such as passwords, are always encrypted. The encryption option cannot be turned off for such data.

Extensible

The message protocol used between the Universal Products components is extensible. New message fields can be added with each new release without creating product component incompatibilities. This permits different component versions to communicate with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without losing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Universal Products programs. The options negotiated include which encryption and compression algorithms are used for the session. Only algorithms that both programs implement are chosen in the negotiation process. The negotiation process permits two different program versions to communicate.

2.5.4 Configurable Attributes

The network protocol can be configured in ways that effect compress, encryption, code pages, and network delays.

The following configuration options are available on many of the Universal Products:

CODE_PAGE

The CODE_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is a text file that contains a two-column table. The table maps local single-byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE_PAGE option value is simply the name of the code page file without any file name extension if present.

CTL_SSL_CIPHER_LIST

The CTL_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most to least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When a manager and server first connect, they perform an SSL handshake. The handshake negotiates the cipher suite used for the session. The manager and server each have a cipher suite list and the first one in common is used for the session.

Why is a list of cipher suites helpful? A distributed software solution may cross many organizational and application boundaries each with their own security requirements. Instead of having to choose one cipher suite for all distributed components, the software components can be configured with their own list of acceptable cipher suites based on their local security requirements. When a high level of security is required, the higher CPU consuming cipher suite is justified. When lower level of security is acceptable, a lower CPU consuming cipher suite may be used. As long as the manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

DATA_AUTHENTICATION

The DATA_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA_AUTHENTICATION option is applicable to the UNVv2 protocol only. SSL always performs authentication.

DATA_COMPRESSION

The DATA_COMPRESSION option specifies that network data should be compressed.

Compression attempts to reduce the amount of data to a form that can be decompressed to its original form. The compression ratio is the original size divided by the compressed size. The compression ratio value will depend on the type of data. Some data compress better than others.

Two methods of compression are available:

- ZLIB method provides the highest compression ratio with the highest use of CPU.
- HASP method provides the lowest compression ratio with the lowest use of CPU.

Whether or not compression is used, and which compression method is used, depends on several items:

- Network bandwidth
If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources
If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio
If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be justified.

DATA_ENCRYPTION

The DATA_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used: SSL or UNVv2.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

DATA_SSL_CIPHER_LIST

The DATA_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See [CTL_SSL_CIPHER_LIST](#) in this section.)

DEFAULT_CIPHER

The DEFAULT_CIPHER option specifies the SSL cipher suite to use (since SSL protocol requires a cipher suite) if the [DATA_ENCRYPTION](#) option is set to NO. The default DEFAULT_CIPHER is NULL-MD5 (no encryption, MD5 message digest).

All SSL cipher suites have a message digest for good reasons. The message digest ensures that the data sent are the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

KEEPALIVE_INTERVAL

The KEEPALIVE_INTERVAL option specifies how often, in seconds, a keepalive message (also commonly known as a heartbeat message) is sent between a manager and server. A keepalive message ensures that the network and both programs are operating normally. Without a keepalive message, error conditions can arise that place one or both programs in an infinite wait.

A keepalive message is sent from the server to the manager. If the server does not receive a keepalive acknowledgement from the manager in a certain period of time (calculated as the maximum of 2 x NETWORK_DELAY or the KEEPALIVE_INTERVAL), the server considers the manager or network as unusable. How the server processes a keepalive time-out depends on what fault tolerant features are being used. If no fault tolerant features are being used, the server ends with an error. The manager expects to receive a keepalive message in a certain period of time (calculated as the KEEPALIVE_INTERVAL + 2 x NETWORK_DELAY).

NETWORK_DELAY

The NETWORK_DELAY option provides the ability to fine tune Universal product's network protocol. When a data packet is sent over a TCP/IP network, the time it takes to reach the other end depends on many factors, such as, network congestion, network bandwidth, and the network media type. If the packet is lost before reaching the other end, the other end may wait indefinitely for the expected data. In order to prevent this situation, Universal Products time out waiting for a packet to arrive in a specified period of time. The delay option specifies this period of time.

NETWORK_DELAY specifies the maximum acceptable delay in transmitting data between two programs. Should a data transmission take longer than the specified delay, the operation ends with a time out error. Universal Products will consider a time out error as a network fault.

The default NETWORK_DELAY value is 120 seconds. This value is reasonable for most networks and operational characteristics. If the value is too small, false network time outs could occur. If the value is too large, programs will wait a long period of time before reporting a time out problem.

SIO_MODE

The SIO_MODE option specifies whether the data transmitted over the network is processed as text data or binary data.

Text data is translated between the remote and local code pages. Additionally, end of line representations are converted

Text translation operates in two modes: direct and UCS. The default is direct. The direct translation mode exchanges code pages between Universal Products components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater than 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation. UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.

2.6 Fault Tolerance

Fault tolerant features address Universal Product capabilities to recover or restart from an array of error conditions that occur in any large IT organization. Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

2.6.1 Network Fault Tolerance

UDM uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of re-sending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs. Like any application using TCP/IP, UDM is subject to these network errors. Should they occur, a product can no longer communicate and must shutdown or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

UDM provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using the network fault tolerant protocol, UDM traps the connection termination caused by the network error and it reestablishes the network connections. Once connections are reestablished, processing automatically resumes from the location of the last successful message exchange. No program restarts are required and no data are lost.

The network fault tolerant protocol acknowledges and checkpoints successfully received and sent messages, respectively. The network fault tolerant protocol does reduce data throughput. Consequentially, the use of network fault tolerance should be carefully weighed in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program than to incur increased execution time.

When a network connection terminates, the manager will enter a network reconnect phase. In the reconnect phase, the manager attempts to connect to the server and reestablish its network connections. The condition that caused the network error may persist for only seconds or days. The manager will attempt server reconnection for a limited amount of time. That amount of time is configured with the `RECONNECT_RETRY_COUNT` and `RECONNECT_RETRY_INTERVAL` options. These two options determine, respectively, how many reconnect attempts are made and how often they are made. After all attempts have failed, the manager ends with an error.

When a network connection terminates, the server enters a disconnected state and waits for the manager to reconnect. The user process continues running; however, if the user process attempts any I/O on the standard files, it will block. The server waits for the manager to reconnect for a period of time defined by the manager's `RECONNECT_RETRY_COUNT` and `RECONNECT_RETRY_INTERVAL`. Once that time has expired, the server terminates the user process and exits.

UDM can request the use of the network fault tolerant protocol. If the server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

The `NETWORK_FAULT_TOLERANT` option is used to request the protocol.

2.6.2 Open Retry

Open Retry is a type of fault tolerance used at the session-establishment level.

(Network fault tolerance, as described in Section [2.6.1 Network Fault Tolerance](#), is used from the time that a session has been fully established until the session has terminated.

Open Retry is used during the establishment phase of a session. UDM tries to establish a session when the `open` command is issued. If the `OPEN_RETRY` option value is **yes**, and UDM fails to establish the session due to a network error, timeout, or the inability to start a transfer server, it will retry the `open` command based on the settings of the `OPEN_RETRY_COUNT` and `OPEN_RETRY_INTERVAL` options.

2.6.3 Component Management

In order to fully understand Universal Data Mover fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component startup, execution, and termination. The broker and its components have the ability to communicate service requests and status information between each other.

The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the broker determines the current state of the component. This state information is required by the broker to determine if a restart or reconnect request from a manager is acceptable or not. The broker's component information can be viewed with the Universal Query program.

One piece of component information maintained by the broker is the component's communication state. The communication state primarily determines what state the Universal Data Mover Server is in regarding its network connection with a manager and the completion of the user process and its associated spooled data.

The communication state values are described in [Table 2.3 Component Communication States](#), below. The Reconnect column indicates whether or not a network reconnect request is valid. The Restart column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
COMPLETED	NO	NO	The server and manager have completed. All standard output and standard error files have been sent to the manager and the user process's exit status.
DISCONNECTED	YES	YES	The server is not connected to the manager. This occurs when a network error has occurred, the manager halted, or the manager host halted. The server is executing with either the network fault tolerant protocol, is restartable, or both. Note: The server cannot tell if the manager is still executing or not since it cannot communicate with it.
ESTABLISHED	NO	NO	The server and manager are connected and processing normally. This state is the most common state when all is well.
RECONNECTING	NO	NO	The server has received a reconnect request from the manager to recover a lost network connection. This state should not remain long, only for the time it takes to re-establish the network connections.
STARTED	NO	NO	The server has started. If the server is restartable it is receiving the standard input file from the manager and spooling it.

Table 2.3 Component Communication States

2.7 z/OS CANCEL Command Support

Universal Products provide network fault tolerance (see Section [2.6 Fault Tolerance](#)) and, in some cases, manager fault tolerance. These features provide users with the ability to execute jobs that will continue to run when the network is down and when a manager is terminated.

However, there are scenarios in which the user may want to cancel an executing job that supports manager and/or network fault tolerance and have both the manager and server processes terminate immediately. Because of fault tolerance, when the manager is terminated, the server side would begin a connection reestablishment protocol and continue to execute. This would allow the started user job to continue running.

In particular, z/OS supports a CANCEL command that will terminate a job executing on the z/OS operating system. When a Universal Data Mover job is cancelled via the z/OS CANCEL command, the job terminates with either of these exit codes:

- Exit code S122, if it is cancelled with a dump.
- Exit code S222, if it is cancelled without a dump.

Part of the responsibility of a Universal Broker executing on a particular host is to monitor the status of all locally running manager processes on that machine. So, when instructed, that Universal Broker could issue a STOP command to the Universal Data Mover Server process associated with the stopped/ended manager process.

In the case of a Universal Data Mover three-party transfer, both the primary and secondary servers need to be cancelled. The Universal Broker running locally with the cancelled Universal Data Mover Manager process will send a STOP command to the primary server. This primary server will, in turn, forward the STOP command to the secondary server, thus cancelling both servers of the three-party transfer.

2.7.1 Exit Codes

Through the use of the [SERVER_STOP_CONDITIONS](#) configuration option, the Universal Data Mover Manager process notifies the locally running Universal Broker of the exit codes that should cause it to terminate the running Server process. With this option, the user can specify a list of exit codes that should trigger the locally running Universal Broker to issue the STOP command to the manager's Universal Data Mover server-side process.

[SERVER_STOP_CONDITIONS](#) can specify a single exit code or a comma-separated list of exit codes. These stop conditions are passed from the manager to the locally running Universal Broker, which store this and other component-specific data about the executing manager component. When this executing Universal Data Mover Manager process is cancelled or stopped, the locally running Universal Broker detects the ending of the manager process and retrieves its process completion information, which includes the exit code of the manager.

The Universal Broker then compares this exit code with the list of exit codes provided by [SERVER_STOP_CONDITIONS](#). If a match is found, and network fault tolerance is enabled, the Universal Broker will execute a uctl command to STOP the running Universal Data Mover Server component.

2.7.2 Security Token

For security purposes, Universal Products pass around a security token that is used by the locally running Universal Broker to STOP associated Universal Data Mover Server process.

This security token is generated on a component-by-component basis by the Universal Broker process that starts the Universal Data Mover Server. Upon generation, this token is returned to the Universal Data Mover Manager which, in turn, updates its locally running Universal Broker with this token. The locally running Universal Broker then uses this token with the issued STOP command to cancel the running Universal Data Mover Server process.

When this token is received by the Universal Broker processes with the request to STOP the server component, the Broker authenticates the received token with the stored token for the running Universal Data Mover Server process. When the token is authenticated, the Universal Data Mover Server process is STOPPED.

2.8 Universal Access Control List

Many Universal Products utilize the Universal Access Control List (UACL) feature as an extra layer of security to the services they offer. The UACL determines if a request is denied or allowed to continue and can assign security attributes to the request.

This section describes the UACL capabilities in general, non-component specific terms. See the appropriate component security sections for complete details on how a component utilizes the UACL feature.

The following Universal Product components use the UACL feature:

- Universal Broker uses UACLs to permit or deny TCP/IP connections based on the remote host IP address.
See the Universal Broker User Guide for complete details.
- Universal Command Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID, and to control whether or not the Manager request requires user authentication.
See the Universal Command 3.2.0 User Guide for complete details.
- Universal Control Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID, and to control whether or not the Manager request requires user authentication.
See the Universal Control chapter of the Universal Products Utilities 3.2.0 User Guide for complete details.
- Universal Data Mover Server uses UACLs to permit or deny Manager access based on the Managers IP address and user ID.
See the UACL section for each operating system in this user guide for complete details.

2.8.1 UACL Configuration

The method used to configure UACL rules is platform dependent. The following sections discuss each of the methods.

z/OS

All UACL rules are defined in library **UNVCONF**, member **ACLCFG00**. The Universal Broker allocates the UACL configuration data set to ddname **UNVACL**.

The UACL file syntax is the same as all other Universal Products z/OS configuration files. See Section [2.2.6 Configuration File Syntax](#) for details.

UNIX

All UACL rules are defined in one file, **uac1.conf**. This file is required for products utilizing UACL rules; otherwise, the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file syntax is the same as all other Universal UNIX configuration files. See Section [2.2.6 Configuration File Syntax](#) for details.

Windows

All UACL rules are stored in the configuration file, **uac1.conf**.

UACL entries for each component are maintained using the Universal Configuration Manager (see Section [2.4 Universal Configuration Manager](#)).

OS/400

All UACL rules are defined in file **unvconf** and member **uac1**. This file is required for products utilizing UACL rules, else the product will not start. The configuration file consists of zero or more UACL entries.

The UACL file is searched for in the same manner as all other product configuration files. See Section [2.2.5 Configuration File](#) for information on how configuration files are located.

The UACL file syntax is the same as all other Universal Products for OS/400 configuration files. See Section [2.2.6 Configuration File Syntax](#) for details.

2.8.2 UACL Entries

UACL entries are composed of two parts: type and rule.

- Type identifies the Universal Products component for which the rule applies. For example, the Universal Broker product utilizes UACL rules of type **ubroker_access**.
- Rule defines the client's identity and the client's request for which the entry pertains and the security attributes it enforces.

UACL configuration file syntax is the same as all other configuration files, where the configuration file keyword corresponds to the UACL type part and the configuration file value corresponds to the UACL rule part.

The entire rule part of the UACL entry must be enclosed in quotation characters, not just a sub-field of the rule, if a space or tab is part of the value.

The correct syntax would be as follows:

```
ucmd_request "prod.host.name,MVS USER,user,cmd,DSPLIB  
QGPL,allow,auth"
```

For each client that connects and sends a request, Broker and Server components search UACL entries to find the best match for the client identity and the client request. Entries are searched in the order they are listed. The first entry found stops the search.

Note: There is no limit to the number of UACL entries that can be specified.

Client Identification

Rule matching is based on the client identity and the client request.

There are two client identification methods:

1. X.509 certificate authentication.
2. Client IP address and reported user account.

X.509 Certificate Authentication

X.509 certificates identify an entity. An entity can be a program, person, or host computer. When an X.509 certificate is authenticated, it authenticates that the entity is who it claims to be.

X.509 certificates are utilized in UACL entries by first mapping a client certificate to a UACL certificate identifier. The certificate identifier then is used in the UACL entries. A certificate identifier provides for:

1. Concise representation of certificates in UACL entries. There are a large number of certificate fields that may be used and many of the fields have lengthy, tedious naming formats. A certificate map only needs to be defined once and then the concise certificate identifier can be used in the UACL entries.
2. Mapping of one or more certificates to a single certificate identity. A group of entities that share a common security access level may be represented by one certificate identity reducing the number of UACL entries to maintain.

UACL certificate map entries are searched sequentially (that is, top to bottom) matching the client certificate to each entry until a match is found. The certificate map defines a set of X.509 certificate fields that may be used as matching criteria.

Table 2.4, below, defines the certificate map matching criteria.

Criteria	Description
SUBJECT	<p>Matches the X.509 <code>subject</code> field. The <code>subject</code> field is formatted as an X.501 Distinguished Name (DN). A DN is a hierarchical list of attributes referred to as Relative Distinguished Names (RDNs).</p> <p>RDNs are separated with a comma (,) by default. If a different separator is required (perhaps one of the RDN values uses a comma), start the DN with the different separator character. Valid separators are slash (/), comma (,) and period (.).</p> <p>Many RDN values can be used in a DN. Some of the most common values are:</p> <ul style="list-style-type: none"> • C Country name • CN Common name • L Locality • O Organization • OU Organizational Unit • ST State <p>The RDN attributes must be listed in the same order as they are defined in the certificate to be considered matched.</p> <p>A partial DN can be specified. All certificates that have a <code>subject</code> name that matches up to the last RDN are considered a match. This permits a group of certificates to be matched.</p> <p>The RDN attribute values can include pattern matching characters. An asterisk (*) matches 0 or more characters and a question mark (?) matches one character.</p> <p>Some example of SUBJECT values are:</p> <ul style="list-style-type: none"> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road Runner"</code> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road *"</code> • <code>subject="C=US,ST=Georgia,O=Acme,CN=Road ?unner"</code> <p>Whether an RDN value is case sensitive or not depends on the format in which the value is stored. The certificate creator has some control over which format is used. All formats except for <code>printableString</code> are case sensitive.</p>

Criteria	Description
EMAIL	<p>Matches the X.509 emailAddress attribute of the subject field and rfc822Name of the subjectAltName extension value. Both fields format the email address as an RFC 822 addr-spec in the form of identifier@domain.</p> <p>The attribute values may include pattern matching characters. An asterisk (*) matches 0 or more characters and a question mark (?) matches one character.</p> <p>Some example EMAIL values are:</p> <ul style="list-style-type: none"> • email=user1@acme.com • email=*@acme.com • email=user?@acme.com <p>RFC 822 names are not case sensitive.</p>
HOSTNAME	<p>Matches the following X.509 fields in the order listed:</p> <ol style="list-style-type: none"> 1. dNSName of the subjectAltName extension value. 2. commonName (CN) RDN attribute of the subject field's DN value. <p>Some example HOSTNAME values are:</p> <ul style="list-style-type: none"> • hostname=bigfish.acme.com • hostname=*.acme.com <p>The values are not case sensitive.</p>
IP ADDRESS	<p>Matches the X.509 iPAddress field of the subjectAltName extension value.</p> <p>An example IPADDRESS value is:</p> <ul style="list-style-type: none"> • ipaddress=10.20.30.40
SERIAL NUMBER	<p>Matches the X.509 serialNumber value.</p> <p>The value can be specified in a hexadecimal format by prefixing the value with 0x or 0X, otherwise, the value is considered a decimal format. For example, the value 0x016A392E7F would be considered a hexadecimal format.</p> <p>An example SERIALNUMBER value is:</p> <ul style="list-style-type: none"> • serialnumber=0x7a2d52cbae

Table 2.4 Certificate Map Matching Criteria

If a certificate map rule is found that matches the client certificate, the rule's identifier is assigned to the client's request. The certificate identifier is then used in matching certificate-based UACL entries.

[Table 2.5](#), below, defines the certificate identifier field as used in UACL entries.

Criteria	Description
CERTID	<p>Matches the certificate identifier defined by the certificate map entry. The CERTID value has the following syntax:</p> <ul style="list-style-type: none"> • An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM, but not ABCDM. • The comparison is case insensitive. • Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, A / *B matches A*B. A / /B matches A/B.

Table 2.5 Certificate Identifier Field

Client IP Address Identification

TCP/IP provides a method to obtain a client's IP address. The IP address typically identifies the host computer on which the client is executing. There are exceptions to this though. Networks can be configured with Network Address Translation (NAT) systems between the client and the Broker that hides the client's IP address. In addition to the client IP address, Universal Products clients provide a user account name with which they are executing that is used to further refine the client's identity.

UACL entries are searched matching the client's IP address and user account to each entry until a match is found.

Table 2.6, below, defined possible matching criteria for IP address and user account client identification.

Criteria	Description
HOST	<p>Matches the TCP/IP address of the remote user.</p> <p>The HOST value has the following syntax:</p> <ul style="list-style-type: none"> Dotted numeric form of an IP address. For example, 10.20.30.40. Dotted numeric prefix of the IP addresses. For example, 10.20.30. matches all IP addresses starting with 10.20.30. The last dot (.) is required. A net/mask expression. For example, 131.155.72.0/255.255.254.0 matches IP address range 131.155.72.0 through 131.155.73.255. The mask and the host value are AND'ed together. The result must match net. <p>Note: Contact your network administrator for calculation of the correct net / mask expression.</p> <ul style="list-style-type: none"> Host name for an IP address. For example, sysa.abc.com. Host name suffix for a range of IP addresses. For example, .abc.com matches all host names ending with abc.com, such as, sysa.abc.com. The first dot (.) is required. A value of ALL matches all IP addresses. The value must be uppercase.
REMOTE_USER	<p>Matches the user name with which the remote user is executing as on the remote system.</p> <p>The REMOTE_USER value has the following syntax:</p> <ul style="list-style-type: none"> An asterisk (*) matches 0 or more characters and a question mark (?) matches one character. For example, AB*M matches ABCDM and ABM. AB?M matches ABCM, but not ABCDM. Control code /c switches off case-sensitivity and /C switches on case-sensitivity matching. The default is <i>on</i>. For example, /cABC matches abc. /ca/Cbc matches Abc, but not ABC. Pattern matching characters, such as the asterisk and question mark, are included in the text to be matched by prefixing them with a forward slash (/) character. For example, A/*B matches A*B. A//B matches A/B.

Table 2.6 Client IP Address - Matching Criteria

Certificate-Based and Non Certificate-Based UACL Entries

Universal Products components that support X.509 certificates define their UACL entries in two varieties:

1. Certificate-based entries
2. Non certificate-based entries

The two entry types are distinguished by their name. For example, `cmd_cert_access` is the certificate-based form of the entry and `ucmd_access` is a non certificate-based entry . All entries follow the same format.

Certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate that matches a certificate map entry.

Non certificate-based UACL entries are searched under the following conditions:

- Client provides an X.509 certificate and no certificate map entry matches.
- Client does not provide an X.509 certificate.

Either the certificate-based UACL entries or the non certificate-based UACL entries are searched, but not both.

2.8.3 Types of UACL Rules

There are three types of ACL rules in the UACL configuration file used by UDM:

- `udm_access`
- `udm_cert_access`
- `udm_mgr_access`

All of them are access rules. That is, they are used to either allow or deny access (the right to establish a connection) from a remote system.

The `udm_access` and `udm_mgr_access` rules are similar to the Universal Command `ucmd_access` rule in that they allow or deny access depending on a remote host IP address and remote user.

`udm_access`

`udm_access` takes the same form as the `ucmd_access` rule. The remote host and remote user refer to the host IP of the machine connecting to the UDM Server and the user of the remote system.

`udm_mgr_access`

`udm_mgr_access` differs slightly from `ucmd_access` in that instead of the remote IP and remote user, `udm_mgr_access` refers to the IP address and user of the manager.

Note: In a three-party transfer session, the manager's host IP is from the perspective of the primary transfer server (the IP address given if you look up the manager's IP address on the primary server).

This also is true for the secondary server. Even if the secondary server, when looking up the manager, would produce a different IP address than the primary server, the ACL rule is based on the address as seen from the primary server.

`udm_cert_access`

`udm_cert_access` works just like the Universal Command `ucmd_cert_access` rule. In the top section of the UACL configuration file, you describe a certificate in detail and give it an alias. A `udm_cert_access` rule describes whether access is allowed or denied to a connecting UDM manager or server based on whether or not its certificate matches the one to which the alias in the rule refers.

For detailed information on these UACL rules, see [Chapter 5 Universal Data Mover UACL Entries](#) in the Universal Data Mover 3.2.0 Reference Guide.

2.8.4 Proxy Certificates

For two-party transfer sessions, certificates work exactly as they do with Universal Command: the manager presents its certificate when trying to establish a session with the server. If the access rules allow a connection using that certificate, and if the certificate is properly validated, a session is established.

For three-party transfer sessions, the manager — as it does in a two-party transfer session — presents its certificate to the primary when trying to establish a session. The `udm_cert_access` rules (if there are any) determine whether the session can be established between the manager and primary server using this certificate.

A similar thing happens when setting up the remainder of the three-party transfer session between the primary and secondary. There are some differences, though. A UDM server cannot have a certificate of its own, so the primary server in a three-party transfer session obtains its certificate from the broker (in effect, it uses the Broker's certificate) running on that machine to try to establish a connection with the secondary. If certificate access rules in the UACL configuration file on the secondary are used to allow or deny this connection with the secondary, they must refer to the Broker's certificate on the primary.

(See Section [1.3 Transfer Sessions](#) for further information on two-party and three-party transfer sessions.)

In order to help minimize the amount of certificate management and set-up that needs to be done with a large number of systems, UDM provides alternative way of using certificates in a three-party transfer session: proxy certificates.

A proxy certificate is a certificate generated on the fly that is signed by an original certificate holder and allows the holder of the proxy certificate to act on behalf of (as the proxy of) the original certificate holder.

In a three-party UDM session, this removes the need to use the Broker's certificate on the primary when establishing a connection from the primary to the secondary. Instead, once the manager's certificate is validated on the primary (just as happens when not using proxy certificates in a three-party transfer session), a proxy certificate is generated by the primary and signed by the manager. The primary uses this proxy certificate to establish a connection between it and the secondary.

The **subject** name of the proxy certificate is the same as the original certificate supplied to the manager. A certificate ACL rule for a certificate containing this subject can be used on both the primary and secondary. Proxy certificates simplify things by allowing the same ACL rule to be used in both places instead having a rule for the manager's certificate on the primary and a rule for the primary's (actually, the broker on the primary system) certificate on the secondary.

Proxy certificates can only be used under the following conditions:

1. Manager, primary, and secondary must all be version 3.2 or later.
2. Manager, primary, and secondary must all use OpenSSL as their SSL type. If any of these are running on the mainframe, they cannot use system SSL and proxy certificates at the same time.
3. `PROXY_CERTIFICATES` configuration option must be set to **yes** for the manager.

2.9 Message and Audit Facilities

All Universal Products have the same message facilities. Messages - in this context - are text messages written to a console, file, or system log that:

1. Document the actions taken by a program.
2. Inform users of error conditions encountered by a program.

This section describes the message and audit facilities that are common to all Universal Products. (See the individual Universal Product documentation for additional details.)

2.9.1 Message Types

There are six types (or severity levels) of Universal Products messages. (The severity level is based on the type of information provided by those messages.)

1. Audit messages document the configuration options used by the program's execution and resource allocation details. They provide complete description of the program execution for auditing and problem resolution.
2. Informational messages document the actions being taken by a program. They help determine the current stage of processing for a program. Informational messages also document statistics about data processed.
3. Warning messages document unexpected behavior that may cause or indicate a problem.
4. Error messages document program errors. They provide diagnostic data to help identify the cause of the problem.
5. Diagnostic messages document diagnostic information for problem resolution.
6. Alert messages document a notification that a communications issue, which does not disrupt the program or require action, has occurred.

The MESSAGE_LEVEL configuration option in each Universal Product component lets you specify which messages are written (see [Section 2.9.3 Message Levels](#)).

2.9.2 Message ID

Each message is prefixed with a message ID that identifies the message.

The message ID format is **UNVnnnn1**, where:

- **nnnn** is the message number.
- **1** is the message severity level:
 - **A** (Audit)
 - **I** (Informational)
 - **W** (Warning)
 - **E** (Error)
 - **T** (alerT)
 - **D** (Diagnostic)

Note: The Universal Products 3.2.0 Messages and Codes document identifies all messages numerically, by product, using the **nnnn** message number.

2.9.3 Message Levels

Each Universal Product includes a **MESSAGE_LEVEL** configuration option that lets you select which levels (that is, severity levels) of messages are to be written.

- *Audit* specifies that all audit, informational, warning, and error messages are to be written.
- *Informational* specifies that all informational, warning, and error messages are to be written.
- *Warning* specifies that all warning and error messages are to be written.
- *Error* specifies that all error messages are to be written.
- *Trace* specifies that a trace file is created, to which data used for program analysis will be written. The trace file name and location are Universal Product dependent (see the appropriate Universal Product documentation for details).
(Trace should be used only at the request of Stonebranch, Inc. [Customer Support](#).)

Note: Diagnostic and Alert messages always are written, regardless of the level selected in the **MESSAGE_LEVEL** option.

2.9.4 Message Destinations

The location to which messages are written is the message destination.

Some Universal Products have a MESSAGE_DESTINATION configuration option that specifies the message destination. If a program is used only from the command line or batch job, it may have only one message destination, such as standard error. Valid destination values will depend on the host operating system.

z/OS

Universal Products on z/OS run as batch jobs or started tasks. Batch jobs do not provide the MESSAGE_DESTINATION option. All messages are written to the SYSOUT ddname.

Started task message destinations are listed in the table below.

Destination	Description
LOGFILE	Messages are written to ddname UNVLOG. All messages written to log files include a date and time stamp and the program's USS process ID.
SYSTEM	Messages are written to the console log as WTO messages.

UNIX

Message destinations are listed in the table below.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. The recommended directory for log files is <code>/var/opt/universal/log</code> . This can be changed with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the syslog daemon. Not all programs provide this destination. Universal programs that execute as daemons write to the syslog's daemon facility. All messages include the programs process ID. If an error occurs writing to the syslog, the message is written to the system console.

Windows

Message destinations are listed in the table below.

Destination	Description
STDERR	Messages are written to standard error. This destination is most useful for console commands.
LOGFILE	Messages are written to a log file. Not all programs provide this destination. Log files are written to product specific log directories, which can be modified with the LOG_DIRECTORY option. All messages written to log files include a date and time stamp and the program's process ID.
SYSTEM	Messages are written to the Windows Application Event Log.

OS/400

Message destinations are listed in the table below.

Destination	Description
STDERR	Messages are written to standard error. A batch job's standard error file is allocated to the print file QPRINT.
LOGFILE	Messages are written to the job's job log.
SYSTEM	Messages are written to the system operator message queue QSYSOPR.

2.10 X.509 Certificates

A certificate is an electronic object that identifies an entity. It is analogous to a passport in that it must be issued by a party that is trusted by all who accept the certificate.

Certificates are issued by trusted parties called Certificate Authorities (CA's). For example, VeriSign Inc. is a CA that most parties trust. We all have faith that a trusted CA takes the necessary steps to confirm the identity of a user before issuing the user a certificate.

Certificate technology is based on public/private key technology. There are a few different types of public/private keys: RSA, DH, and DSS. As their name denotes, the private key must be kept private, like a password. The public key can be given to anyone or even published in a newspaper.

A property of public/private keys is that data encrypted with one can be decrypted only with the other. Therefore, if someone wants to send you a secret message, they encrypt the data with your public key, which everyone has. However, since you are the only one with your private key, you are the only one who can decrypt it. If you want to send someone message, such as a request for \$100,000 purchase, you can "sign" it with your private key.

Note: Signing does not encrypt the data. Once a person receives your request, that person can verify it is from you by verifying your electronic signature with your public key.

A certificate ties a statement of identity to a public key. Without the public key, the certificate is meaningless. Possession of a certificate alone does not prove your identity. You must have the corresponding private key. The two together prove your identity to any third party that trusts the CA that issued your certificate. This is a key point; if you do not trust the CA that signed a certificate, you cannot trust the certificate.

Since certificates originally were designed to be used for internet authentication, global directory technologies were developed to make them available via the internet. This directory technology is known as X.500 Directory Access Protocol. Later LDAP was introduced by Netscape to make it Lightweight Directory Access Protocol.

X.500 divides the world into a hierarchical directory. A person's identity is located by traversing down the hierarchy until it reaches the last node. Each node in the hierarchy consists of a type of object, such as a country, state, company, department, or name.

2.10.1 Sample Certificate Directory

Figure 2.7, below, provides a sample diagram of a small X.500 directory.

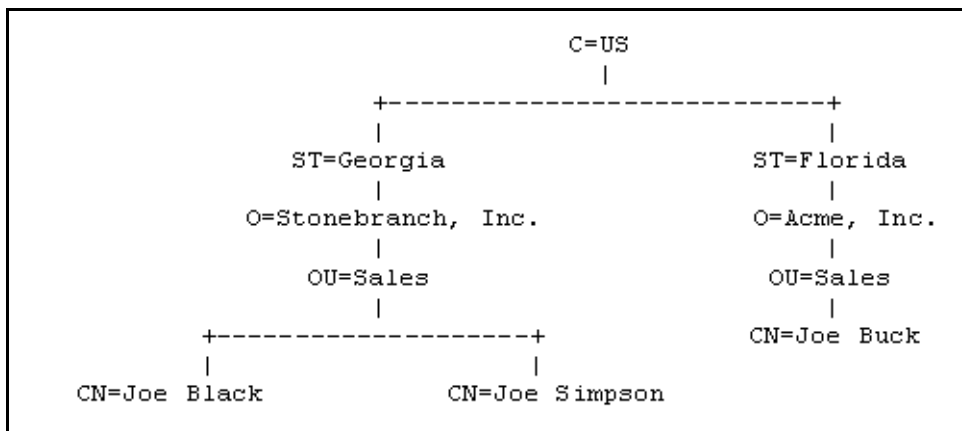


Figure 2.7 X.500 Directory (sample)

The keywords listed on each node are referred to as a Relative Distinguished Name (RDN). A person is identified by a Distinguished Name (DN). The DN value for Joe Black is **C=US/ST=Georgia/O=Stonebranch, Inc./OU=Sales/CN=Joe Black**.

A certificate is composed of many fields and possible extensions. Many of the most popular fields are specified as X.500 DN values.

2.10.2 Sample X.509 Certificate

Figure 2.8, below, illustrates a sample X.509 version 3 certificate for Joe Buck at the Acme corporation.

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      01:02:03:04:05:06:07:08
    Signature Algorithm: md5withRSAEncryption
    Issuer: C=US, ST=Florida, O=Acme, Inc., OU=Security, CN=CA
    Authority/emailAddress=ca@acme.com
    Validity
      Not Before: Aug 20 12:59:55 2004 GMT
      Not After : Aug 20 12:59:55 2005 GMT
    Subject: C=US, ST=Florida, O=Acme, Inc., OU=Sales, CN=Joe Buck
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:be:5e:6e:f8:2c:c7:8c:07:7e:f0:ab:a5:12:db:
          fc:5a:1e:27:ba:49:b0:2c:e1:cb:4b:05:f2:23:09:
          77:13:75:57:08:29:45:29:d0:db:8c:06:4b:c3:10:
          88:e1:ba:5e:6f:1e:c0:2e:42:82:2b:e4:fa:ba:bc:
          45:e9:98:f8:e9:00:84:60:53:a6:11:2e:18:39:6e:
          ad:76:3e:75:8d:1e:b1:b2:1e:07:97:7f:49:31:35:
          25:55:0a:28:11:20:a6:7d:85:76:f7:9f:c4:66:90:
          e6:2d:ce:73:45:66:be:56:aa:ee:93:ae:10:f9:ba:
          24:fe:38:d0:f0:23:d7:a1:3b
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Subject Alternative Name:
        email:joe.buck@acme.com
    Signature Algorithm: md5withRSAEncryption
    a0:94:ca:f4:d5:4f:2d:da:a8:6d:e3:41:6e:51:83:57:b3:b5:
    31:95:32:b6:ca:7e:d1:4f:fb:01:82:db:23:a0:39:d8:69:71:
    31:9c:0a:3b:ce:f6:c6:e2:5c:af:23:f0:d7:ee:87:3e:8a:7b:
    40:03:39:64:a1:8c:29:7d:5b:99:93:fa:23:19:e1:e4:ac:4d:
    13:0f:de:ad:51:27:e3:4e:4b:9f:40:4c:05:fd:f2:82:09:3e:
    46:05:f0:ad:cc:f7:78:25:3e:11:f8:ca:b6:df:f7:37:57:9b:
    63:00:d0:b5:b5:18:ec:38:73:d2:85:a3:c7:24:21:47:ee:f2:
    8c:0d
  
```

Figure 2.8 X.509 Version 3 Certificate (sample)

Note: The contents of a certificate file does not look like the information in [Figure 2.8](#), which is produced by a certificate utility using the certificate file as input. Certificates can be saved in multiple file formats, so their file contents will look very different.

Certificate Fields

A certificate is composed of many fields.

[Table 2.7](#), below, describes the main fields.

Field or Section	Description
Version	X.509 certificates come in two versions: 1 and 3.
Serial Number	CA is required to provide each certificate it issues a unique serial number. The serial number is not unique for all certificates, only for the certificates issued by each CA.
Issuer	DN name of the CA that issued the certificate.
Validity	Starting and ending date for which this certificate is valid.
Subject	Identity of the certificate. A certificate may identify a person or a computer. In this case, the certificate identifies Joe Buck in the Sales organization of the Acme company in the state of Florida in the United States.
Public Key	Public key associated with the certificate identity.
X509v3 Extensions	X.509 version 3 introduced this section so that additional certificate fields may be added. In this case, the identity's email address is included as a Subject Alternative Name field. This section is not available in X.509 version 1.
Signature	CA's digital signature of the certificate.

Table 2.7 Certificate Fields

2.10.3 SSL Peer Authentication

The SSL protocol utilizes X.509 certificates to perform peer authentication. For example, a Universal Command Manager may want to authenticate that it is connected to the correct Broker.

Peer authentication is performed by either one or both of the programs involved in the network session. If a Manager wishes to authenticate the Broker to which it connects, the Broker will send its certificate to the Manager for the Manager to authenticate. Should the Broker wish to authenticate the Manager, the Manager sends its certificate to the Broker.

Certificate authentication is performed in the following steps:

1. Check that the peer certificate is issued by a trusted CA.
2. Check that the certificate has not been revoked by the CA.
3. Check that the certificate identifies the intended peer.

If a step fails, the network session is terminated immediately.

Certificate Verification

The Universal Product must be configured with a list of trusted CA certificates. When a peer certificate is received, the trusted CA certificates are used to verify that the peer certificate is issued by one of the trusted CA's.

The trusted CA certificate list must be properly secured so that only authorized accounts have update access to the list. Should the trusted CA list become compromised, there is a possibility that an untrusted CA certificate was added to the list.

The CA certificate list configuration option is `CA_CERTIFICATES`. It specifies a PEM formatted file that contains one or more CA certificates used for verification.

Should a peer certificate not be signed by a trusted CA, the session is immediately terminated.

Certificate Revocation

After a certificate is verified to have come from a trusted CA, the next step is to check if the CA has revoked the certificate. Since a certificate is held by the entity for which it identifies, a CA cannot take a certificate back after it is issued. So when a CA needs to revoke a certificate for some reason, it issues a list of revoked certificates referred to as the Certificate Revocation List (CRL). A program that validates certificates needs to have access to the latest CRL issued by the CA.

The `CERTIFICATE_REVOCATION_LIST` configuration option specifies the PEM-formatted file that contains the CRL. This option is available in all Universal Products that utilize certificates.

Certificate Identification

When a certificate has been validated as being issued by a trusted CA, and not revoked by the CA, the next step is to check that it identifies the intended peer.

A Universal Product Manager validates a Broker certificate by the Broker host name or IP address or the certificate serial number. The `VERIFY_HOST_NAME` configuration option is used to specify the host name or IP address that is identified in the Broker certificate. Each certificate signed by a CA must have a unique serial number for that CA. The `VERIFY_SERIAL_NUMBER` option is used to specify the serial number in the Broker certificate.

Should certificate identification fail, the session is immediately terminated.

Universal Brokers work differently than the Managers. A Broker maps a peer certificate to a certificate ID. The certificate map definitions are part of the Universal Access Control List (UACL) definitions. At that point, the certificate ID is used by UACL definitions to control access to Broker and Server services.

Certificate Support

Many certificate authority applications, also known as Public Key Infrastructure (PKI) applications, are available. Universal Products should be able to utilize any certificate in a PEM format file. PEM (Privacy Enhanced Mail) is a common text file format used for certificates, private keys, and CA lists.

Universal Products support X.509 version 1 and version 3 certificates.

Although implementing a full featured PKI infrastructure is beyond the scope of Universal Products and this documentation, some assistance is provided using the OpenSSL toolkit (<http://www.openssl.org>).

Universal Products on most of the supported platforms utilize the OpenSSL toolkit for its SSL and certificate implementation. OpenSSL is delivered on most UNIX distributions and Windows distributions are available on the OpenSSL web site.

Universal Products supports z/OS System SSL on the IBM z/OS operating system as well as OpenSSL. System SSL interfaces directly with the RACF security product for certificate access. All certificates, CA and user certificates, and private keys must be stored in the RACF database to use System SSL.

The Universal Product suite includes an X.509 certificate utility, Universal Certificate, to create certificates for use in the Universal Product suite. See the Universal Certificate chapter in the Universal Products Utilities 3.2.0 User Guide for details.

Chapter 3

Universal Data Mover Manager for z/OS

3.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the z/OS operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

The z/OS Batch Manager provides a batch job interface to remote computers running the UDM Server component.

3.2 Usage

UDM Manager for z/OS executes as a batch job.

This section describes the JCL, configuration and configuration options, and command line syntax of UDM Manager for z/OS.

Section [3.3 Examples of UDM Manager for z/OS](#) provides examples demonstrating the flexibility of Universal Data Mover.

3.2.1 JCL Procedure

Figure 3.1, below, illustrates the Universal Data Mover for z/OS JCL procedure (UDMPRC, located in the **SUNVSAMP** library) that is provided to simplify the execution JCL and future maintenance.

```
//UDMPRC  PROC  UPARM=,                -- UDM options
//              USPRFC=USPRFC00,        -- USAP SAP RFC member
//              UNVPRE=#SHLQ.UNV,
//              UNVPRD=#PHLQ.UNV
//*
//PS1      EXEC  PGM=UDM,REGION=256M,PARM=' ENVAR(TZ=EST5EDT)/&UPARM'
//STEPLIB  DD   DISP=SHR,DSN=&UNVPRE..SUNVLOAD
//*
//UNVNLS   DD   DISP=SHR,DSN=&UNVPRE..SUNVNLS
//UNVUSRC   DD  DISP=SHR,DSN=&UNVPRD..UNVCONF(&USPRFC)
//UNVCLIB   DD  DISP=SHR,DSN=&UNVPRE..SUNVSAMP
//*
//UNVTRACE DD   SYSOUT=*
//UNVTRMDL DD  DISP=SHR,DSN=&UNVPRD..MDL
//*
//SYSPRINT DD   SYSOUT=*
//SYSOUT    DD   SYSOUT=*
//CEEDUMP   DD   SYSOUT=*
//SYSUDUMP  DD   SYSOUT=*
//*
//SYSIN     DD   DUMMY                -- UDM command options
//UNVSCR     DD   DUMMY                -- UDM script
```

Figure 3.1 UDM Manager for z/OS – JCL Procedure

For this JCL procedure:

- UPARM parameter is used to specify EXEC PARM keyword values.
- UNVPRE parameter specifies the data set name prefix of Universal Products installation data sets.
- UNVPRD parameter specifies the data set name prefix of Universal Products production data sets.

3.2.2 DD Statements in JCL

Table 3.1, below, describes the DD statements used in the Universal Data Mover for z/OS JCL illustrated in Figure 3.1.

ddname	DCB Attributes	Mode	Description
STEPLIB	DSORG=PO, RECFM=U	input	Universal Products load library containing the program being executed.
UNVNLS	DSORG=PO, RECFM=(F, FB, V, VB)	input	Universal Products national language support library. Contains message catalogs and code page translation tables.
UNVUSRC	DSORG=PS, RECFM=(F, FB, V, VB)	input	Universal SAP connector RFC member.
UNVCLIB	DSORG=PO, RECFM=(F, FB, V, VB)	input	UDM call library: UDM searches for script files specified on the call command in this library.
UNVTRACE	DSORG=PS, RECFM=(F, FB, V, VB)	Output	UDM trace output.
UNVTRMDL	DSORG=PS, RECFM=(F, FB, V, FB)	input	Data set used as a model for creating UCMD and USAP trace files when they are called by UDM using the exec and execsap commands.
SYSPRINT	DSORG=PS, RECFM=(F, FB, V, VB)	output	Standard output file for the UDM program.
SYSOUT	DSORG=PS, RECFM=(F, FB, V, VB)	output	Standard error file for the UDM program.
SYSIN	DSORG=PS, RECFM=(F, FB, V, VB)	input	Standard input file for the UDM program. UDM reads its command options from SYSIN.
UNVSCR	DSORG=PS, RECFM=(F, FB, V, VB)	input	UDM command script: UDM executes the script allocated to this ddname.
The C runtime library determines the default DCB attributes. Refer to the IBM manual <i>OS/390 C/C++ Programming Guide</i> for details on default DCB attributes for stream I/O.			

Table 3.1 UDM Manager for z/OS – DD Statements in JCL

3.2.3 JCL

[Figure 3.2](#), below, illustrates the Universal Data Mover for z/OS JCL using the **UDMPRC** procedure illustrated in [Figure 3.1](#).

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1   EXEC UDMPRC
//UNVSCR  DD  *
open srv=sol7 user=id001 pwd=pwd001
copy local='uid.data' srv=data
quit
/*
```

Figure 3.2 UDM Manager for z/OS – JCL

Job step STEP1 executes the procedure **UDMPRC**.

The UDM script commands are specified on the **UNVSCR DD**.

3.2.4 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. PARM keyword
2. SYSIN ddname
3. Configuration file

The order of precedence is the same as the list above; PARM keyword options being the highest and configuration file being the lowest. That is, options specified via a PARM keyword override options specified via a SYSIN ddname, and so on.

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

See [Section 2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

3.2.5 Configuration Options

[Table 3.2](#), below, describes the configuration options used to execute UDM Manager for z/OS.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
ALLOC_ABNORMAL_DISP	Abnormal disposition of a data set being allocated.
ALLOC_BLKSIZE	Block size used for newly allocated data sets.
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.
ALLOC_DSORG	Data set organization used for newly allocated data sets.
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.
ALLOC_LRECL	Logical record length used for newly allocated data sets.
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.
ALLOC_NORMAL_DISP	Normal disposition of a data set being allocated.
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.
ALLOC_RECFCM	Record format used for newly allocated data sets.
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.
ALLOC_UNIT	Unit used for newly allocated data sets.
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.

Option Name	Description
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PRIVATE_KEY	ddname of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SAF_KEY_RING	SAF certificate key ring name.
SAF_KEY_RING_LABEL	SAF key ring certificate label.
SCRIPT	ddname from which to read a UDM script command file.

Option Name	Description
SCRIPT_OPTIONS	Options to pass to the script command file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SERVER_STOP_CONDITIONS	Exit codes that cause Universal Broker to cancel the corresponding UDM Server of the exited UDM Manager.
SSL_IMPLEMENTATION	SSL implementation.
SYSTEM_ID	Local Universal Broker with which the UDM Manager must register.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 3.2 UDM Manager for z/OS - Configuration Options

3.2.6 Command Line Syntax

Figure 3.3 and Figure 3.4, below, illustrate the command line syntax of UDM Manager for z/OS.

```
udm
[-alloc_abnormal_disp {keep|delete|catlg|uncatlg}]
[-alloc_blksize size]
[-alloc_dataclas class]
[-alloc_dir_blocks number]
[-alloc_dsorg {po|ps}]
[-alloc_input_status {old|shr}]
[-alloc_lrecl length]
[-alloc_mgmtclas class]
[-alloc_normal_disp {keep|delete|catlg|uncatlg}]
[-alloc_output_status {old|shr}]
[-alloc_prim_space space]
[-alloc_recfm format]
[-alloc_sec_space space]
[-alloc_space_unit {cyl|trk|number}]
[-alloc_storclas class]
[-alloc_unit unit]
[-alloc_volser number]
[-system_id ID]
[-ssl_implementation {openssl|system}]
[-ca_certs ddname]
[-cert ddname]
[-private_key ddname [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl ddname]
[-script ddname]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-compress {yes|no}[,{zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[,{time|notime}] ]
```

Figure 3.3 UDM Manager for z/OS - Command Line Syntax (1 of 2)

```
[-network_fault_tolerant {yes|no} [-frame_interval number] ]  
[-mode_type {binary|text}]  
[-umask number]  
[-outboundip host]  
[-port port]  
[-recvbuffersize size]  
[-open_retry {yes|no}]  
[-open_retry_count number]  
[-open_retry_interval number]  
[-retry_count number]  
[-retry_interval seconds]  
[-sendbuffersize size]  
[-saf_key_ring name]  
[-saf_key_ring_label label]  
[-server_stop_conditions codes]  
[-tcp_no_delay option]  
[-tracefilelines number]  
[-trace_table size, {error|always|never}]  
[-comment text]  
  
udm  
{-help | -version}
```

Figure 3.4 UDM Manager for z/OS - Command Line Syntax (2 of 2)

3.3 Examples of UDM Manager for z/OS

[Appendix A Examples](#) provides operating system-specific examples that demonstrate the use of Universal Data Mover.

Included in this appendix are the following examples that demonstrate the use of Universal Data Mover Manager for z/OS:

- [Copy a File to an Existing z/OS Sequential Data Set](#)
- [Copy a z/OS Sequential Data Set to a File](#)
- [Copy a Set of Files to an Existing z/OS Partitioned Data Set](#)
- [Copy a File to a New z/OS Sequential Data Set](#)
- [Copy a Set of Files to a New z/OS Partitioned Data Set](#)

For each topic, there is an example (as appropriate) for both the DSN and DD file systems.

Note: These z/OS examples apply equally as well to the Windows operating systems, with appropriate changes for the file system syntactical differences.

3.4 Security

Universal Data Mover is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files
2. Privacy and integrity of transmitted network data

3.4.1 Data Set Permissions

Only trusted user accounts should have write access to the Universal Data Mover installation files. Eligible users of Universal Data Mover require read access to the national language support library **SUNVNLS**, the configuration file **UNVCONF**, and the load library **SUNVLOAD**.

Chapter 4

Universal Data Mover Manager for Windows

4.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the Windows operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

4.2 Usage

The UDM Manager command is executed from the command line or a script. After it has been initiated, UDM is ready to process commands. The commands can come from standard input or a script file.

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for Windows.

Section [4.3 Examples of UDM Manager for Windows](#) provides examples demonstrating the flexibility of Universal Data Mover.

4.2.1 Modes of Operation

Under Windows, UDM can be run either in:

- Interactive mode
- Batch mode

Running UDM in Interactive Mode

To invoke UDM in interactive mode, enter the following at the command prompt:

```
udm
```

This will start the UDM Manager. You will be greeted with a startup message and the UDM prompt, similar to this:

```
UNV2800I Universal Data Mover 3.2.0 Level 0 started.  
udm>
```

At the **udm>** prompt, you can enter any UDM command.

To exit UDM, enter the following command at the **udm>** prompt:

```
quit
```

Running UDM in Batch Mode

When running in batch mode, UDM processes a script file.

When the script file has finished executing, UDM will exit automatically:

```
udm -s script_filename
```

4.2.2 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

Although configuration files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to set configuration options. The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values (see [Section 2.4 Universal Configuration Manager](#)).

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

See [Section 2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

4.2.3 Configuration Options

[Table 4.1](#), below, describes the configuration options used to execute UDM Manager for Windows.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
INSTALLATION_DIRECTORY	Directory on which UDM Manager is installed.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.

Option Name	Description
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PRIVATE_KEY	ddname of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Options to pass to the script command file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UCMD_PATH	Sets the complete path to UCMD for calls by the exec command.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 4.1 UDM Manager for Windows - Configuration Options

4.2.4 Command Line Syntax

Figure 4.1, below, illustrates the command line syntax of UDM Manager for Windows.

```
udm
[-ca_certs file]
[-cert file]
[-private_key file [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl file]
[-script filename]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-compress {yes|no}[, {zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[, {time|notime}] ]
[-network_fault_tolerant {yes|no} [-frame_interval number] ]
[-mode_type {binary|text}]
[-umask number]
[-outboundip host]
[-port port]
[-recvbuffersize size]
[-open_retry {yes|no}]
[-open_retry_count number]
[-open_retry_interval number]
[-retry_count number]
[-retry_interval seconds]
[-sendbuffersize size]
[-tcp_no_delay option]
[-tracefilelines number]
[-trace_table size, {error|always|never}]
[-comment text]

udm
{-help | -version}
```

Figure 4.1 UDM Manager for Windows - Command Line Syntax

4.3 Examples of UDM Manager for Windows

[Appendix A Examples](#) provides operating system-specific examples that demonstrate the use of Universal Data Mover.

Included in this appendix are the following examples that demonstrate the use of Universal Data Mover Manager for Windows (and UNIX):

- [Simple File Copy to the Manager](#)
- [Simple File Copy to the Server](#)
- [Copy a Set of Files](#)

Each example illustrates a procedure that occurs under the operating system's default file system.

(See Section [A.2 UDM Manager for z/OS Examples](#) in [Appendix A Examples](#) for z/OS examples that apply equally as well to the Windows operating systems.)

Chapter 5

Universal Data Mover Manager

for UNIX

5.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the UNIX operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

5.2 Usage

The UDM Manager command is executed from the command line or a script. After it has been initiated, UDM is ready to process commands. The commands can come from standard input or a script file.

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for UNIX.

Section [5.3 Examples of UDM Manager for UNIX](#) provides examples demonstrating the flexibility of Universal Data Mover.

5.2.1 Modes of Operation

Under UNIX, UDM can be run either in:

- Interactive mode
- Batch mode

Running UDM in Interactive Mode

To invoke UDM in interactive mode, enter the following at the command prompt:

```
udm
```

This will start the UDM Manager. You will be greeted with a start-up message and the UDM prompt, similar to this:

```
UNV2800I Universal Data Mover 3.2.0 Level 0 started.  
udm>
```

At the **udm>** prompt, you can enter any UDM command.

To exit UDM, enter the following command at the **udm>** prompt:

```
quit
```

Running UDM in Batch Mode

UDM also can be run in batch mode. When running in batch mode, UDM processes a script file. When the script file has finished executing, UDM will exit automatically:

```
udm -s script_filename
```

5.2.2 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

The configuration file, `udm.conf`, provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources. (See the Universal Products 3.2.0 Installation Guide to determine the directory in which it is located.)

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

5.2.3 Configuration Options

[Table 5.1](#), below, describes the configuration options used to execute UDM Manager for UNIX.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
BIF_DIRECTORY	Broker Interface File directory where the Universal Broker interface file is located.
CA_CERTIFICATES	File name / ddname of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name / ddname of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name / ddname of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
HELP	Writes a description of the command options and their format.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
INSTALLATION_DIRECTORY	Directory in which Universal Data Mover is installed.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.

Option Name	Description
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OPEN_RETRY	Level of fault tolerance for the open command.
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the open command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PLF_DIRECTORY	Program Lock File directory that specifies the location of the UDM Manager program lock file.
PRIVATE_KEY	ddname of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party transfer sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Parameters to pass to the script file.
SEND_BUFFER_SIZE	Size (in bytes) of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UCMD_PATH	Sets the complete path to UCMD for calls by the exec command.
UMASK	File mode creation mask.
USAP_PATH	Sets the complete path to USAP for calls by the execsap command.
VERSION	Writes the program version information and copyright.

Table 5.1 UDM Manager for UNIX - Configuration Options

5.2.4 Command Line Syntax

Figure 5.1, below, illustrates the command line syntax of UDM Manager for UNIX.

```
udm
[-bif_directory directory]
[-plf_directory directory]
[-ca_certs file]
[-cert file]
[-private_key file [-private_key_pwd password] ]
[-proxy_certificates {yes|no}]
[-crl file]
[-script filename]
[-options options]
[-codepage codepage]
[-ctl_ssl_cipher_list list]
[-data_ssl_cipher_list list]
[-compress {yes|no}[, {zlib|hasp}] ]
[-delay seconds]
[-idle_timeout seconds]
[-keep_alive_interval seconds]
[-lang language]
[-level {trace|audit|info|warn|error}[, {time|notime}] ]
[-network_fault_tolerant {yes|no} [-frame_interval number] ]
[-mode_type {binary|text}]
[-umask number]
[-outboundip host]
[-port port]
[-recvbuffersize size]
[-open_retry {yes|no}]
[-open_retry_count number]
[-open_retry_interval number]
[-retry_count number]
[-retry_interval seconds]
[-sendbuffersize size]
[-tcp_no_delay option]
[-tracefilelines number]
[-trace_table size, {error|always|never}]
[-comment text]

udm
{-help | -version}
```

Figure 5.1 UDM Manager for UNIX - Command Line Syntax

5.3 Examples of UDM Manager for UNIX

[Appendix A Examples](#) provides operating system-specific examples that demonstrate the use of Universal Data Mover.

Included in this appendix are the following examples that demonstrate the use of Universal Data Mover Manager for Windows (and UNIX):

- [Simple File Copy to the Manager](#)
- [Simple File Copy to the Server](#)
- [Copy a Set of Files](#)

Each example illustrates a procedure that occurs under the operating system's default file system.

5.4 Security

Universal Data Mover is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files and directories
2. Access to Universal Data Mover configuration files
3. Universal Data Mover user account
4. Privacy and integrity of transmitted network data
5. User authentication

5.4.1 File Permissions ---

Only trusted user accounts should have permission to write to the Universal Data Mover installation directory and subdirectories, and all files within those directories.

5.4.2 Configuration Files ---

Only trusted user accounts should have write permission to the Universal Data Mover configuration files, and add and delete access to the directories in which they reside.

Chapter 6

Universal Data Mover Manager

for OS/400

6.1 Overview

This chapter provides information on the Universal Data Mover (UDM) Manager, specific to the OS/400 operating system.

UDM Manager transfers files between any computers running UDM Server. Using a UDM command script, you indicate to the UDM Manager the actions to take. The UDM Manager connects to the UDM Server (or Servers) and processes your request.

6.2 Usage

The UDM Manager command is invoked from a command line or a batch job. After it has been initiated, UDM is ready to process commands.

The commands can come from:

- Standard input or a script file in interactive mode
- Script file in batch mode

This section describes the modes of operation, configuration and configuration options, and command line syntax of UDM Manager for UNIX.

Section [6.3 Examples of UDM Manager for OS/400](#) provides examples demonstrating the flexibility of Universal Data Mover.

6.2.1 Universal Products for OS/400 Commands ---

The names of the Universal Products for OS/400 commands that are installed in the OS/400 **QSYS** library are tagged with the Universal Products for OS/400 **version / release / modification number, 320**. The names of the commands installed in the Universal Products for OS/400 product library, **UNVPRD320**, are untagged.

To maintain consistency across releases, you may prefer to use the untagged names in your production environment. The **UCHGRLS** (Change Release Tag) program lets you change the tagged command names in **QSYS** to the untagged command names in **UNVPRD320**.

(See the Universal Products 3.2.0 for OS/400 Installation Guide for detailed information on **UCHGRLS**.)

This chapter references the OS/400 commands by their untagged names. If you are using commands with tagged names to run UDM, substitute the tagged names for the untagged names in these references.

6.2.2 Modes of Operation

Under OS/400, UDM can be run either:

- In interactive mode
- From a script file
- As a batch application

Additionally under OS/400, UDM can use either the LIB or HFS file system.

Running UDM Interactively

To invoke UDM as an interactive application:

1. Enter the following on the command line: **STRU DM**
2. Press <Enter>.

This will start the UDM Manager. You will be greeted with a startup message and the **udm>** prompt, similar to this:

```
UNV2800I Universal Data Mover 3.2.0 Level 0 started.
```

```
udm>
```

At the **udm>** prompt, you can enter any UDM command.

To exit UDM, enter the following command at the **udm>** prompt:

```
quit
```

Running UDM from a Script

To invoke a UDM script:

1. Enter the following on the command line:
STRU DM SCRFILE(library/file) SCRMBR(member)
2. Press <Enter>.

This will start the UDM Manager using the script located by library, file, and member. When the script file has finished executing, UDM will exit automatically.

UDM requires the member name; there is no default. Requiring the member name makes script specification under OS/400 behave as it does on other platforms. On other systems, there is no default search order as exists under OS/400. However, users may explicitly provide ***FILE** as a member name to use the OS/400 default file search order.

Running UDM in Batch Mode

UDM also can be run in batch mode. When running in batch mode, use a script as shown in Section [Running UDM from a Script](#).

To execute a batch file such as the one below, use:

SBMDBJOB FILE(LIBNAME/FILENAME) MBR(MBRNAME):

```
//BCHJOB JOB(MYUDMJOB) ENDSEV(10)
STRUDM MYLIB/QSCRSRC UDM817
//ENDBCHJOB
```

Output is sent to the output queue associated with the batch job. Two spooled files will be sent to the output queue; one file associated with standard out and one file associated with standard error.

6.2.3 Configuration

Configuration consists of:

- Setting default options and preferences for all executions of UDM Manager.
- Setting options and preferences for a single execution of UDM Manager.

Configuration options are read from the following sources:

1. Command line
2. Command file
3. Environment variables
4. Configuration file

The order of precedence is the same as the list above; command line options being the highest and configuration file being the lowest. That is, options specified via the command line override options specified via a command file, and so on.

The configuration file provides the simplest method of specifying configuration options whose values will not change with each command invocation. These default values are used if the options are not read from one or more other sources.

The installation default for the UDM configuration file is Universal Products installation library **UNVPRD320**, file **UNVCONF**, and member **UDM**. The configuration file name can be any valid file name. It can be edited manually using an OS/400 editor such as SEU, EDTF, or any other installed source file editor, or via the IFS using a text editor. If a text editor is used to edit the file via the IFS, the padded spaced must be removed for lines that exceed the file maximum record length.

Some options only can be specified in the configuration file; they have no corresponding command line equivalent. Other options cannot be specified in the configuration file; they must be specified via one or more other sources for a single execution of UDM Manager.

See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

6.2.4 Configuration Options

[Table 6.1](#), below, describes the configuration options used to execute UDM Manager for OS/400.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CA_CERTIFICATES	File name of the PEM-formatted trusted CA X.509 certificates.
CERTIFICATE	File name of UDM Manager's PEM-formatted X.509 certificate.
CERTIFICATE_REVOCATION_LIST	File name of the PEM-formatted CRL.
CODE_PAGE	Character code page used to translate text data received and transmitted over the network.
CODEPAGE_TO_CCSID_MAP	Specification to use either the internal or external table for code page to CCSID mapping.
COMMENT	User-defined string.
CTL_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the control session between UDM components.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	Acceptable and preferred SSL cipher suites to use for the data session on which file data is transferred between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent when UDM is operating with network fault tolerance turned on.
IDLE_TIMEOUT	Number of seconds of inactivity in an interactive UDM session after which the manager will close the session.
KEEP_ALIVE_INTERVAL	Default interval at which a keep alive message is sent from the manager to the transfer server(s).
MERGE_LOG	Specification for whether or not to merge standard out and standard error output streams from a remote command to the UDM transaction log.
MESSAGE_LANGUAGE	Universal Message Catalog (UMC) file used to write messages.
MESSAGE_LEVEL	Level of messages to write.
MODE_TYPE	Default transfer mode type for UDM sessions.
NETWORK_DELAY	Expected network latency.
NETWORK_FAULT_TOLERANT	Specification for whether or not UDM transfer sessions are network fault tolerant by default.
OPEN_RETRY	Level of fault tolerance for the open command.

Option Name	Description
OPEN_RETRY_COUNT	Maximum number of attempts that will be made to establish a session by the <code>open</code> command.
OPEN_RETRY_INTERVAL	Number of seconds that UDM will wait between each open retry attempt.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
PLF_DIRECTORY	Program Lock File directory that specifies the location of the UDM Manager program lock file.
PRIVATE_KEY	Location of Manager's PEM formatted RSA private key.
PRIVATE_KEY_PWD	Password for the Manager's PRIVATE_KEY.
PROXY_CERTIFICATES	Specification for whether or not UDM will use proxy certificates in three-party sessions if a certificate is supplied to the UDM Manager.
RECONNECT_RETRY_COUNT	Number of attempts the manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
REMOTE_PORT	TCP port number on the remote computer used for invoking UDM Server instances.
SCRIPT_FILE	Script file containing UDM commands to execute.
SCRIPT_OPTIONS	Parameters to pass to the script file.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SIZE_ATTRIB	Default file creation size for physical files of both data and source file types.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
VERSION	Writes the program version information and copyright.

Table 6.1 UDM Manager for OS/400 - Configuration Options

6.2.5 Command Line Syntax

Figure 6.1, below, illustrates the command line syntax of UDM Manager for OS/400.

```

STRUDM
[SCRFILE([library/]filename) [SCRMBR(member) ]
[PLFDIR (directory)]
[OPTIONS(options)]
[CODEPAGE(codepage)]
[CTLCPHRLST(cipherlist)]
[DTACPHRLST(cipherlist)]
[COMPRESS(*{yes|no})]
[CMPSMTH(*{zlib|hasp})]
[DELAY(seconds)]
[IDLTIMOUT(seconds)]
[KEEPLIVE(seconds)]
[MODETYPE (*{bin|binary|text})]
[MSGLANG(language)]
[MSGLEVEL(*{trace|audit|info|warn|error}) [, *{yes|no} ]
[NETWORKFT(*{yes|no}) [FRAMEINT(number)]]
[OPENRETRY(*{yes|no} count interval)]
[OUTBOUNDIP(host)]
[PORT(port)]
[PROXYCERT(option)]
[RCVBUFSIZE(size)]
[RETRYCNT(number)]
[RETRYINT(seconds)]
[SNDBUFSIZE(size)]
[TRCLINES(number)]
[TRCTBL(size, *{error|always|never})]
[CACERTS(file [lib] ) [CACERTSMBR(member) ]
[CERT(file [lib] ) [CERTMBR(member)]
    PVTKEYF(file [lib] ) [PVTKEYFMBR(member)] [PVTKEYPWD(password)] ]
[CRLFILE(file [lib]) [CRLMBR(member)]]
[COMMENT(user-defined string)]

STRUDM
[VERSION(*{yes|no})]

```

Figure 6.1 UDM Manager for OS/400 - Command Line Syntax

6.3 Examples of UDM Manager for OS/400

[Appendix A Examples](#) provides operating system-specific examples that demonstrate the use of Universal Data Mover.

Included in this appendix are the following examples that demonstrate the use of Universal Data Mover Manager in a two-party mode between OS/400 and UNIX:

- [Copy a File to an Existing OS/400 File](#)
- [Copy an OS/400 Data Physical File to a File](#)
- [Copy a Set of Files to an Existing Data Physical File](#)
- [Copy a File to a New OS/400 Data Physical File](#)
- [Copy a File to a New OS/400 Source Physical File](#)
- [Copy a Set of Files to a New Data Physical File on OS/400](#)
- [Copy Different Types of OS/400 Files using forfiles and \\$\(_file.type\)](#)
- [Invoke a Script from a Batch Job](#)

Note: These examples apply equally as well to the Windows operating system, with appropriate changes for the file system syntactical differences.

Each topic provides an example for the LIB file system.

The first topic, [Copy a File to an Existing OS/400 File](#), also provides an example specific to the HFS file system. For other examples similar to those used in the HFS file system, see Section [A.3 UDM Manager for UNIX and Windows Examples](#) in [Appendix A Examples](#).

These examples reference the OS/400 commands by their untagged names. If you are using commands with tagged names to run UDM, substitute the tagged names for the untagged names. (See Section [6.2.1 Universal Products for OS/400 Commands](#) for further information.)

6.4 Security

Universal Data Mover is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files and libraries.
2. Access to the Universal Data Mover configuration file.
3. Privacy and integrity of transmitted network data.

6.4.1 Object Permissions ---

Only administrator accounts should have write permission to the following Universal Products libraries (and all objects within these libraries):

- Installation library, **UNVPRD320** (by default)
- Product temporary library, **UNVTMP320**
- Universal spool library, **UNVSPL320**

For maximum security, only trusted accounts (administrators and the **UNVUBR320** profile) should have management, existence, alter, add, update, and delete authority to these objects.

Note: System value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

Chapter 7

Universal Data Mover Server

for z/OS

7.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the z/OS operating system:

- [Component Definition](#)
- [Configuration](#)
- [Security](#)

7.2 Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

The UDM Server for z/OS component definition is located in the component definition library `#HLQ.UNV.UNVCOMP` allocated to the Universal Broker ddname `UNVCOMP`. The UDM Server component definition member is `UDSCMP00`.

[Table 7.1](#), below, identifies all of the options that comprise the UDM Server for z/OS component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Member name of the UDM Server configuration file in the <code>UNVCONF</code> library allocated to the Broker ddname <code>UNVCONF</code> .
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Member name of the UDM Server program.
WORKING_DIRECTORY	HMS directory used as the working directory of the UDM Server.

Table 7.1 UDM Server for z/OS - Component Definition Options

7.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values. See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

7.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The UDM Server configuration file name is specified in the UDM Server component definition. The default name is **UDSCFG00**. The name refers to a member in the PDS allocated to the Universal Broker ddname UNVCONF.

7.3.2 Configuration Options

Figure 7.2, below, identifies all UDM Server for z/OS command options.

Each **Option Name** is a link to detailed information about that command option in the Universal Data Mover 3.2.0 Reference Guide.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
ALLOC_ABNORMAL_DISP	Abnormal disposition of a data set being allocated.
ALLOC_BLKSIZE	Block size used for newly allocated data sets.
ALLOC_DATACLAS	SMS data class used for newly allocated data sets.
ALLOC_DIR_BLOCKS	Number of directory blocks for newly allocated partitioned data sets.
ALLOC_DSORG	Data set organization used for newly allocated data sets.
ALLOC_INPUT_STATUS	Status of data sets being allocated for input.
ALLOC_LRECL	Logical record length used for newly allocated data sets.
ALLOC_MGMTCLAS	SMS management class used for newly allocated data sets.
ALLOC_NORMAL_DISP	Normal disposition of a data set being allocated.
ALLOC_OUTPUT_STATUS	Status of existing data sets being allocated for output.
ALLOC_PRIM_SPACE	Primary space allocation used for newly allocated data sets.
ALLOC_RECFCM	Record format used for newly allocated data sets.
ALLOC_SEC_SPACE	Secondary space allocation used for newly allocated data sets.
ALLOC_SPACE_UNIT	Space unit in which space is allocated for newly allocated data sets.
ALLOC_STORCLAS	SMS storage class used for newly allocated data sets.
ALLOC_UNIT	Unit used for newly allocated data sets.
ALLOC_VOLSER	Volume serial number used for newly allocated data sets.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.

Option	Description
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 7.2 UDM Server for z/OS - Configuration Options

7.4 Security

Universal Data Mover Server security concerns are:

1. Access to product data sets
2. Access to Universal Product configuration files
3. Universal Broker user account
4. Privacy and integrity of transmitted network data
5. User authentication

7.4.1 File Permissions

Only trusted user accounts should have write permission to the Universal Data Mover Server installation data sets. No general user access is required.

7.4.2 Configuration Files

Only trusted user accounts should have write permission to the Universal Data Mover Server configuration files.

7.4.3 Universal Data Mover Server User ID

Universal Data Mover Server requires read access to its installation data sets and its HFS working directory (defined in the component definition).

7.4.4 User Authentication

User authentication is the process of verifying that a user is known and valid to the system. The process used by UDM Server requires the user to provide a user name / ID and a password. The UDM Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

7.4.5 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains Universal Data Mover Server entries that contain Access Control List (ACL) rules that permit or deny access to the Server.

See Section [2.8 Universal Access Control List](#) for details on the Universal Access Control List feature.

UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

[Table 7.3](#) identifies all UDM Server for z/OS UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Data Mover 3.2.0 Reference Guide.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 7.3 UDM Server for z/OS - UACL Entries

UACL Examples

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access    10.20.30.,*,*,allow,auth
udm_access    ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access    10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access    10.20.30.40,TS1004,*,allow,auth
udm_access    10.20.30.40,*,*,deny,auth
udm_access    ALL,*,root,deny,auth
```

Chapter 8

Universal Data Mover Server for Windows

8.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the Windows operating system:

- [Component Definition](#)
- [Configuration](#)
- [Security](#)

8.2 Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

Although component definition files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to edit component definitions for Windows (see Section [2.4 Universal Configuration Manager](#)).

Note: The component definitions for all Universal Products are identified in the Component Definitions property page of the Universal Broker (see [Figure 8.1](#), below).

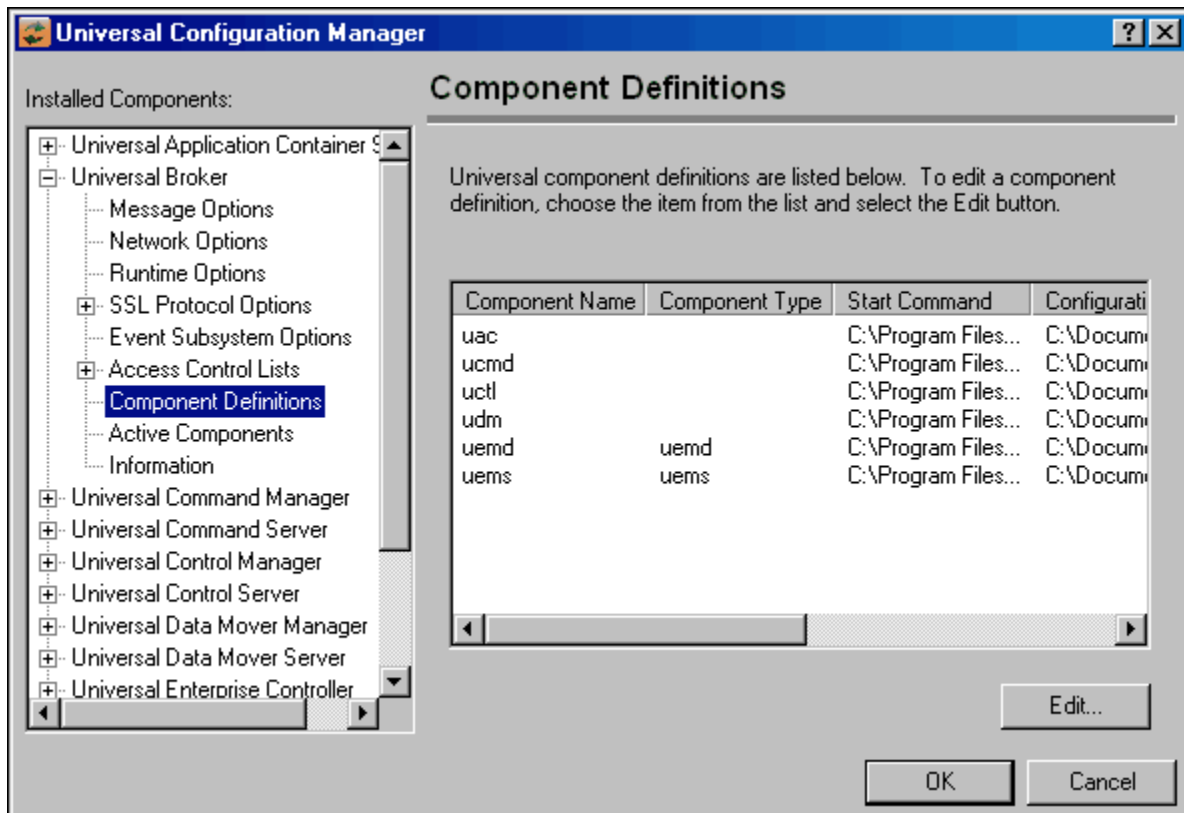


Figure 8.1 Universal Configuration Manager - Component Definitions

The UDM Server component definition is located in the component definition directory of the Universal Broker.

[Table 8.1](#), below, identifies all of the options that comprise the UDM for Windows component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 8.1 UDM Server for Windows - Component Definition Options

8.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values. This section describes the UDM Server configuration options.

See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

8.3.1 Configuration File

The configuration file provides a method of specifying configuration values that will not change with each command invocation.

The Universal Data Mover Server configuration file name (and directory) is specified in the Universal Data Mover Server component definition (see [Chapter 4 Universal Data Mover Component Definition Options](#) in the Universal Data Mover 3.2.0 Reference Guide). The default configuration file name is `udms.conf`.

Although configuration files can be edited manually with any text editor (for example, Notepad), the Universal Configuration Manager application is the recommended way to set configuration options in the configuration file.

The Universal Configuration Manager provides a graphical interface and context-sensitive help, and helps protect the integrity of the configuration file by validating all changes to configuration option values (see Section [2.4 Universal Configuration Manager](#)).

8.3.2 Configuration Options

Table 8.2 identifies all Universal Data Mover Server for Windows configuration options.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
INSTALLATION_DIRECTORY	Directory on which UDM Server is installed.
LOGON_METHOD	Specification for how users are logged onto the system.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_DIRECTORY	Directory name that UDM Server uses for its Trace files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
USER_SECURITY	User security option.

Table 8.2 UDM Server for Windows - Configuration Options

8.4 Security

UDM Server is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. UDM Server has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

UDM security concerns are:

1. Access to Universal Data Mover files and directories.
2. Access to Universal Data Mover configuration files.
3. Universal Data Mover user account.
4. Privacy and integrity of transmitted network data.
5. User authentication.

8.4.1 File Permissions

Only trusted user accounts should have write permission to the UDM Server installation directory and subdirectories, and all of the files within them.

8.4.2 Configuration Files

Only trusted user accounts should have write permission to the UDM Server configuration files, and add and delete access to the directories in which they reside.

8.4.3 Universal Data Mover Server User ID

UDM Server requires read access to its installation directory and its working directory (defined in the component definition).

8.4.4 User Authentication

User authentication is the process of verifying that a user is known and valid to the system. The process used by UDM Server requires the user to provide a user name / ID and a password. The UDM Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

For Windows, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. (With security enabled, you transfer files using a specific user's security context.)

8.4.5 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

See Section [2.8 Universal Access Control List](#) for details on the Universal Access Control List feature.

UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

[Table 8.3](#) identifies all Universal Data Mover Server for Windows UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Data Mover 3.2.0 Reference Guide.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 8.3 UDM Server for Windows - UACL Entries

Updating the Universal Data Mover Server ACL Entries

Although UACL files can be edited with any text editor (for example, Notepad), the Universal Configuration Manager application, accessible via the Control Panel, is the recommended way to update UACL entries (see Section [2.4 Universal Configuration Manager](#)). From there, ACL entries can be added, changed, deleted or sorted (rules are applied in the order in which they are listed).

[Figure 8.2](#), below, illustrates an example. The set of ACL entries only allows connections from host 10.20.30.40 if the user on that host is TS1004. All other remote users will be blocked. TS1004 may run processes on the local system using any user account, provided the correct password is supplied. No processes may be run with Universal Data Mover using the Administrator account on the local system, regardless of where the request originated.

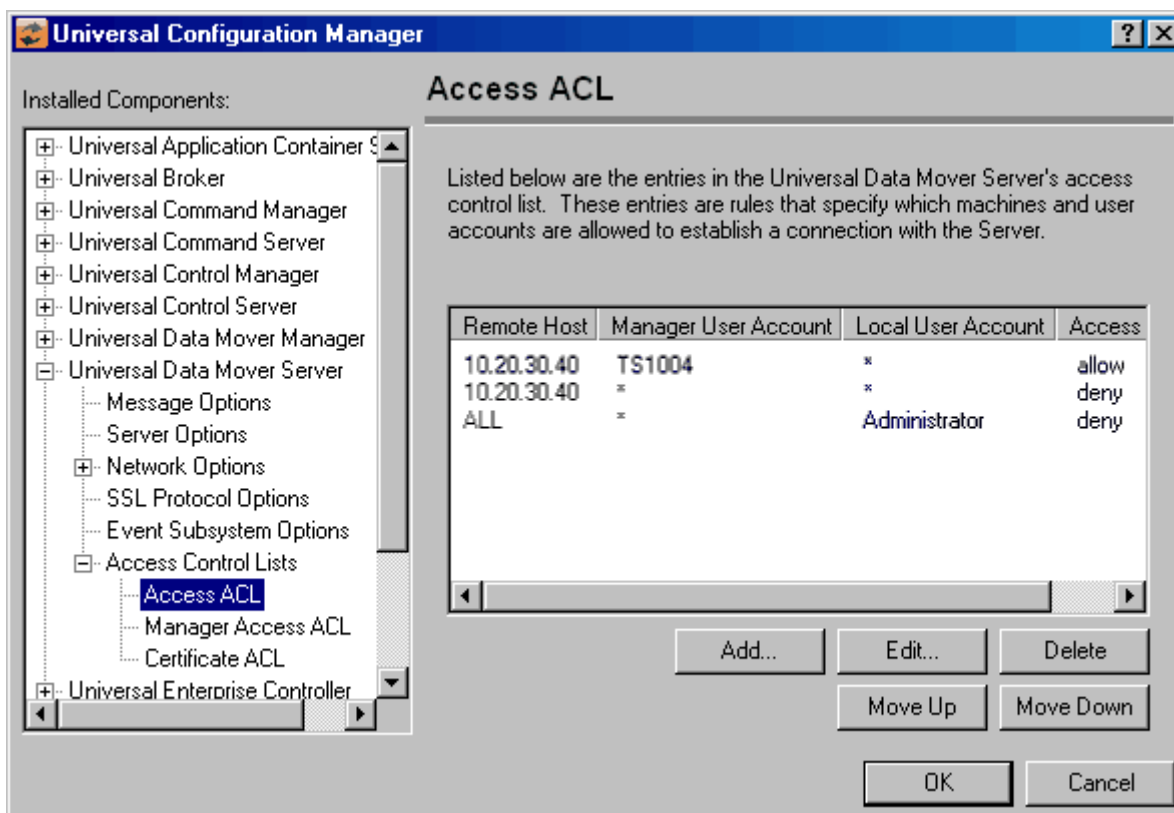


Figure 8.2 Universal Configuration Manager - Universal Data Mover Server - Access ACL

Chapter 9

Universal Data Mover Server

for UNIX

9.1 Overview

This chapter provides the following information on Universal Data Mover (UDM) Server, specific to the UNIX operating system:

- [Component Definition](#)
- [Configuration](#)
- [Security](#)

9.2 Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

The UDM Server for UNIX component definition is located in the component definition directory of the Universal Broker.

[Table 9.1](#), below, identifies all of the options that comprise the UDM Server for UNIX component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 9.1 UDM Server for UNIX - Component Definition Options

9.3 Configuration

Universal Data Mover Server configuration consists of defining runtime and default values. This section describes the UDM Server configuration options.

See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

9.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The Universal Data Mover Server configuration file name (and directory) is specified in the Universal Data Mover Server component definition (see [Chapter 4 Universal Data Mover Component Definition Options](#) in the Universal Data Mover 3.2.0 Reference Guide). The default configuration file name is `udms.conf`.

This file can be edited manually with any text editor.

9.3.2 Configuration Options

[Table 9.2](#) identifies all UDM Server for UNIX configuration options.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
INSTALLATION_DIRECTORY	Directory on which UDM is installed.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
NLS_DIRECTORY	Directory where the UDM Manager message catalog and code page tables are located.
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
TCP_NO_DELAY	Specification for whether or not to use TCP packet coalescing.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_DIRECTORY	Directory that UDM Server uses for its Trace files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 9.2 UDM Server for UNIX - Configuration Options

9.4 Security

Universal Data Mover Server is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover Server has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files and directories
2. Access to Universal Data Mover configuration files
3. Universal Data Mover user account
4. Privacy and integrity of transmitted network data
5. User authentication

9.4.1 File Permissions

Only trusted user accounts should have write permission to the Universal Data Mover Server installation directory and subdirectories, and all of the files within them.

9.4.2 Configuration Files

Only trusted user accounts should have write permission to the Universal Data Mover Server configuration files, and add and delete access to the directories in which they reside.

9.4.3 Universal Data Mover Server User ID

Universal Data Mover Server requires read access to its installation directory and its working directory (defined in the component definition). If user security is activated, the Server requires root access to create processes that execute with another user's identity. The Server security identity is inherited from the Broker. If the Broker is running with a non-root user ID, then the Server program must have the set user ID on execution permission set and root as owner.

9.4.4 User Authentication

User authentication is the process of verifying that a user is known and valid to the system. The process used by UDM Server requires the user to provide a user name / ID and a password. The UDM Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

For UNIX, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. With security enabled, you transfer files using a specific user's security context.

Universal Data Mover can use three different types of user authentication methods:

1. Default authentication uses the UNIX traditional password comparison method.
2. PAM authentication uses the PAM API to authenticate users. The PAM modules, which authenticate and account, are called. This option is available only for certain UNIX platforms.
3. HP-UX Trusted Security uses HP-UX Trust Security APIs to authenticate users. This is available only on Hewlett Packard HP-UX platforms.

9.4.5 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

See Section [2.8 Universal Access Control List](#) for details on the Universal Access Control List feature.

UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

[Table 9.3](#) identifies all UDM Server for UNIX UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Data Mover 3.2.0 Reference Guide.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 9.3 UDM Server for UNIX - UACL Entries

UACL Examples

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access    10.20.30.,*,*,allow,auth
udm_access    ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access    10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access    10.20.30.40,TS1004,*,allow,auth
udm_access    10.20.30.40,*,*,deny,auth
udm_access    ALL,*,root,deny,auth
```

Chapter 10

Universal Data Mover Server for OS/400

10.1 Overview

This chapter provides the following information on the Universal Data Mover (UDM) Server, specific to the OS/400 operating system:

- [Component Definition](#)
- [Configuration](#)
- [Security](#)

10.2 Component Definition

All Universal Products components managed by Universal Broker have a component definition. The component definition is a text file of options containing component-specific information required by Universal Broker. (For details on how Universal Broker manages components, see the Universal Broker 3.2.0 User Guide.)

The syntax of a component definition file is the same as a configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

The UDM Server for OS/400 component definition is located in the component definition directory of the Universal Broker.

[Table 10.1](#), below, identifies all of the options that comprise the UDM Server for OS/400 component definition.

Each **Option Name** is a link to detailed information about that component definition option in the Universal Data Mover 3.2.0 Reference Guide.

Option Name	Description
AUTOMATICALLY_START	Specification for whether or not UDM Server starts automatically when Universal Broker is started.
COMPONENT_NAME	Name by which the clients know the UDM Server.
CONFIGURATION_FILE	Full path name of the UDM Server configuration file.
RUNNING_MAXIMUM	Maximum number of UDM Servers that can run simultaneously.
START_COMMAND	Full path name of the UDM Server program.
WORKING_DIRECTORY	Full path name of the UDM Server working directory.

Table 10.1 UDM Server for OS/400 - Component Definition Options

10.3 Configuration

UDM Server configuration consists of defining runtime and default values. This section describes the Server configuration options.

See Section [2.2.1 Configuration Methods](#) for details on Universal Products configuration methods.

10.3.1 Configuration File

The configuration file provides the simplest method of specifying configuration values that will not change with each command invocation.

The UDM Server configuration file name is specified in the UDM Server component definition (see Section [10.2 Component Definition](#)). The default name is **UNVPRD320/UNVCONF (UDMS)**.

This file can be edited manually with any text editor.

10.3.2 Configuration Options

Table 10.2 identifies all Universal Data Mover Server for OS/400 configuration options.

Each **Option Name** is a link to detailed information about that configuration option in the Universal Data Mover 3.2.0 Reference Guide.

Option	Description
ACTIVITY_MONITORING	Specification for whether or not product activity monitoring events are generated.
CODE_PAGE	Character code page used to translate text data.
CODEPAGE_TO_CCSID_MAP	Specification to use either the internal or external table for code page to CCSID mapping.
DATA_COMPRESSION	Specification for whether or not data is compressed on all standard I/O files.
DATA_SSL_CIPHER_LIST	SSL cipher suites to use for data session between UDM primary and secondary servers.
EVENT_GENERATION	Events to be generated as persistent events.
FRAME_INTERVAL	Number of UDM transfer blocks transferred before a frame-sync message is sent with network fault tolerance turned on.
MESSAGE_LEVEL	Level of messages that UDM will write to the Universal message Catalog (UMC) file.
NETWORK_DELAY	Expected network latency (in seconds).
OUTBOUND_IP	Host or IP address that UDM binds to when initiating outgoing connections to another UDM server.
RECONNECT_RETRY_COUNT	Number of attempts that the UDM Manager will make to re-establish a transfer session when a network fault occurs.
RECONNECT_RETRY_INTERVAL	Number of seconds that UDM will wait between each successive attempt to re-establish a transfer session when a network fault occurs.
RECV_BUFFER_SIZE	Size of the TCP receive buffer for UDM.
SEND_BUFFER_SIZE	Size of the TCP send buffer for UDM.
SIZE_ATTRIB	Default size for file creation of physical files for both data and source file types.
TMP_DIRECTORY	Directory that UDM Server uses for temporary files.
TRACE_FILE_LINES	Maximum number of lines to write to the trace file.
TRACE_TABLE	Size of a wrap-around trace table maintained in memory.
UMASK	File mode creation mask.
USER_SECURITY	User security option.

Table 10.2 UDM Server for OS/400 - Configuration Options

10.4 Security

Universal Data Mover Server is designed to be a secure system. As the level of security rises, so does the administrative complexity of the system. Universal Data Mover Server has balanced the two to avoid the administrative complexity with a minimum sacrifice to security.

Universal Data Mover security concerns are:

1. Access to Universal Data Mover files and libraries
2. Access to Universal Data Mover configuration file
3. Universal Data Mover user account
4. Privacy and integrity of transmitted network data
5. User authentication

10.4.1 Object Permissions ---

Only administrator accounts should have write permission to the following Universal Products libraries (and all objects within these libraries):

- Installation library, **UNVPRD320** (by default)
- Product temporary library, **UNVTMP320**
- Universal spool library, **UNVSPL320**

For maximum security, only trusted accounts (administrators and the **UNVUBR320** user profile) should have management, existence, alter, add, update or delete authority to these objects. As a reminder, the system value **QCRTAUT** controls public access authority to created objects unless overridden by specific commands.

10.4.2 Universal Data Mover Server User Profile ---

If user security is activated, the UDM Server requires, by default, ***ALLOBJ** authority to switch user profiles. As described in Chapter 6 Universal Broker for OS/400 of the Universal Broker 3.2.0 User Guide, this ***ALLOBJ** authority requirement may be removed. The UDM Server initially inherits authority from the **UNVUBR320** user profile. Following the switch to the user profile, the UDM Server runs under the authority of the user initiating the data transfer.

The **UNVUBR320** user profile requires ***SPLCTL** authority in order to provide Universal Submit Job with job logs in specific limited situations. See Chapter 6 Universal Broker for OS/400 in the Universal Broker 3.2.0 User Guide for information on how to remove the ***SPLCTL** authority. Removing ***SPLCTL** from the **UNVUBR320** user profile may prevent the job log processing in limited situations.

10.4.3 User Authentication

User authentication is the process of verifying that a user is known and valid to the system. The process used by UDM Server requires the user to provide a user name / ID and a password. The UDM Server passes the name / ID and password to the operating system for verification; this is referred to as logging on the user.

For OS/400, user authentication is optional. However, if security is enabled, a user name / ID and password are required in order to verify the user's credentials. With security enabled, you transfer files using a specific user's security context.

10.4.4 Universal Access Control List

UDM Server uses the Universal Access Control List (UACL) file as an extra layer of security. The UACL file contains UDM Server entries that contain Access Control List (ACL) rules that permit or deny access to the UDM Server.

See Section [2.8 Universal Access Control List](#) for details on the Universal Access Control List feature.

UACL Entries

The syntax of a UACL entry file is the same as the UDM configuration file. See Section [2.2.6 Configuration File Syntax](#) for detailed syntax information.

[Table 10.3](#) identifies all UDM Server for OS/400 UACL entries.

Each **UACL Entry Name** is a link to detailed information about that UACL entry in the Universal Data Mover 3.2.0 Reference Guide.

UACL Entry Name	Description
UDM_ACCESS	Allows or denies access to Universal Data Mover Server services
UDM_MGR_ACCESS	Allows or denies access based on the host name and/or user of the Manager trying to initiate a UDM session

Table 10.3 UDMr Server for OS/400 - UACL Entries

UACL Examples

The following set of rules permit services for the subnet 10.20.30 and denies all other connections.

```
udm_access    10.20.30.,*,*,allow,auth
udm_access    ALL,*,*,deny,auth
```

The following set of rules effectively permit connections from any host, but has limited access from host 10.20.30.40 to user TS1004 on that host. No host can execute commands as local user root. User TS1004 on host 10.20.30.40 can execute commands as local user tsup1004 without providing the password. Users TS1004 from host 10.20.30.40 can execute commands as any local user by providing the local user password.

```
udm_access    10.20.30.40,TS1004,tsup1004,allow,noauth
udm_access    10.20.30.40,TS1004,*,allow,auth
udm_access    10.20.30.40,*,*,deny,auth
udm_access    ALL,*,root,deny,auth
```

Chapter 11

UDM Scripting Language

11.1 Overview

This chapter provides information on the Universal Data Mover (UDM) scripting language.

UDM has an easy-to-learn scripting language that can be used to give instructions to UDM in both interactive and batch mode. While simple to use, UDM's scripting language has some powerful features, such as the ability to nest script file calls up to ten levels deep, and parameters.

11.2 UDM Commands

The UDM Manager processes commands using UDM's scripting language.

[Table 11.1](#), below, identifies all of the UDM commands.

Each **Command Name** is a link to detailed information about that command in the Universal Data Mover 3.2.0 Reference Guide.

Command Name	Description
appenddata	Appends a line of text to the end of an existing data element, or creates a new data element. containing that line of text.
attrib	Sets the file system attributes that govern the transfer operations on the host with the specified logical name.
break	Stops iterating through a forfiles loop and picks up execution at the script line immediately following the end statement marking the end of the forfiles loop.
call	Loads and executes a command script.
cd	Changes the working directory (on UNIX, Windows, OS/400, and file system HFS) or if z/OS, the current data set qualifiers (for DSN and DD file systems) on the specified logical machine to the specified path.
close	Closes the current transfer session.
closelog	Closes the open log file.
compare	Compares two strings of data.
copy	Initiates a copy operation.
copydir	Initiates a copy operation that recurses into subdirectories.
data	Defines an in-stream data element that can be passed as input for other commands.
debug	Turns debug information on and off.
delete	Deletes a file (or series of files if file-spec contains any wildcards) from the transfer server with the corresponding logical name.
deletestring	Removes a substring from an existing string.
echo	Sends text to standard out (stdout).
echolog	Sends text to an open log file.
exec	Executes system commands on remote machines.
execsap	Executes SAP events.
exit	Exits the UDM Manager (same as the quit command).
filesys	Sets the file system with which the server with the specified logical name is working.
filetype	Set a series of masks and corresponding transfer mode types.
find	Finds a specific occurrence of a substring in an existing string or list element.
format	Creates a formatted string.
insertstring	Inserts a substring into an existing string.

Command Name	Description
<code>loaddata</code>	Loads the contents of a data element from a file, instead of setting them in a script via the data command.
<code>logdata</code>	Writes the content of a data element to the open log file.
<code>lower</code>	Forces all alpha characters in a given variable or list element to lower case.
<code>mode</code>	Sets the current transfer mode.
<code>move</code>	Initiates a move operation.
<code>open</code>	Opens a UDM session.
<code>openlog</code>	Opens a log file on disk for writing custom log information.
<code>pad</code>	Takes a string in an existing variable or list element and pads it to make it the given length.
<code>parse</code>	Parses a string, placing the components of the string into variables.
<code>print</code>	Prints a message in the UDM manager's transaction output.
<code>query</code>	Prints out the UDM Manager version.
<code>quit</code>	Exits the UDM Manager (same as the <code>exit</code> command).
<code>rename</code>	Renames a file.
<code>replace</code>	Replaces one or more instances of a sequence with another sequence.
<code>report</code>	Sets UDM's reporting options.
<code>resetattribs</code>	Resets the attributes for all UDM file systems on the transfer server with the specified logical name.
<code>return</code>	Stops executing the current script immediately and returns execution to the calling script immediately after the <code>call</code> command used to invoke the current script.
<code>reverse</code>	Reverses the order of all characters in the string of a specified existing variable or element.
<code>savedata</code>	Writes each line of a data element to a file on disk.
<code>set</code>	Sets the UDM Manager's built-in and global variable values.
<code>status</code>	Displays the current connection status.
<code>strip</code>	Strips occurrences of a sequence from a string.
<code>substring</code>	Finds a substring in an existing string and stores it in a variable.
<code>truncate</code>	Truncates a string to a specific length.
<code>upper</code>	Forces all alpha characters in a given variable or list element to upper case.

Table 11.1 UDM Commands

11.3 UDM Command Format

All UDM commands conform to the following format:

```
command [parameter_1[=value_1]]...[parameter_n[=value_n]]
```

11.3.1 Basic Rules

The following basic rules apply to all UDM commands.

Parameters

Each command can have zero or more parameters. Each parameter can have a value, which immediately must follow an equal (=) sign.

Spaces

A space must precede each parameter or parameter and value.

Value names, such as a filename with a long path under Windows, can include spaces. To indicate such values, use quotation marks (").

For example:

```
copy src="c:\program files\somefile.txt" dst=test.txt
```

Escape Sequences

Double Quote Marks

To include quotation marks (") as part of the token, use two quotation marks in a row:

```
> echo "This word is ""quoted""!"  
This word is "quoted"!
```

Other Printable Characters

When processing tokens that are inside quotation marks, all other printable characters - except variable references - are ignored as being part of the language.

If you want to assign a variable to have a value of a language symbol, such as an equal sign (=), you must enclose it in quotation marks:

```
> set myvar=="="  
> echo $(myvar)  
=
```

Line Continuation

If a command is too long for a single line, it can be continued on one or more following lines by placing either of the following characters as the last character in each line break:

- Plus sign (+)
Retains leading white space on the next line when assembling the finished line.
- Minus sign (-)
Trims the leading white space.

For example:

```
This is +  
    a test
```

Yields the following line:

```
This is      a test
```

```
This is -  
    a test
```

Yields the following line:

```
This is a test
```

Comments

A script also can have comments: lines of user-specified text indicating information about the script and the operations taking place. Comment lines begin with the hash (#) mark. White space characters can precede the hash (#) mark.

11.3.2 Sample UDM Script

The following is a sample UDM script:

```
# Open a transfer session  
open src=* dst=ntmachine  
  
# Copy command using line continuation  
copy src=test.txt +  
    dst=test.txt  
  
if 8 EQ $( _lastrc )  
    print msg="The last command resulted in an error"  
end  
  
# Close the transfer session and exit UDM  
quit
```

11.3.3 Expressions

The following basic rules apply to expressions in UDM commands.

Appearance

An expression can appear either as a parameter or its value. It must comprise the entire parameter or value in which it appears, not just part of it.

For example, in the following command, `<2 + 2>` is not an expression:

```
echo "2 + 2 = <2 + 2>"
```

It is merely part of the quoted string, and the output would be:

```
2 + 2 = <2 + 2>
```

In order to treat `<2 + 2>` as an expression, the command must be:

```
echo "2 + 2 = " <2 + 2>
```

The output of this command is:

```
2 + 2 = 4
```

Integer Only

Although floating point numbers are allowed in expressions, everything is evaluated as an integer. The only exception is that the [EQ - Equal](#) and [NE - Not Equal](#) comparators can be used to compare strings as well as numbers.

Delimiters

All expressions must be bound by left angle (`<`) and right angle (`>`) brackets.

For example:

```
set value=<2 + 2>
```

Operand / Operator Delimiters

Operands and operators in an expression must be separated by a space.

For example, the following is not legal:

```
<2+4>
```

There must be a space before and after the operator `< + >`:

```
<2 + 4>
```

Operator Precedence

The operator order of precedence (and reading left to right) is:

1. NOT
2. *, /, and %
3. + and -
4. EQ, NE, LT, GT, LE, and GE
5. AND, OR, and XOR

To manually indicate that an operation is of higher precedence, enclose it in parentheses: (and). An expression is evaluated going from the inner most set of parentheses out. Sets on the same level are evaluated left to right.

The following examples illustrates how parentheses can affect results.

In the following expression, where < * > takes precedence over < + >:

```
<5 + 4 * 2>
```

The expression yields the following value:

```
13
```

However, when the expression includes the following parentheses:

```
<(5 + 4) * 2>
```

The expression yields the following value:

```
18
```

Nesting

Expressions can be nested in order to indicate desired change precedence. Nested expressions are bound by parentheses: (and).

When nested expressions are a part of an expression, the deepest nested portions are evaluated first. Also, spaces are not required between the (and) when they are used for nesting, as they are between operators and operands.

For example:

```
> echo <5 + 5 * 2>
```

```
15
```

```
> echo <(5 + 5) * 2>
```

```
20
```

```
> echo <2 * 4 + (2 * (7 + 2))>
```

```
26
```

```
> echo <2 * 4 + 2 * 7 + 2>
```

```
24
```


Operations

All operations take the following form:

left-value operator right-value

The **left-value** and **right-value** can be:

- Strings (EQ and NE only)
- Numbers
- Variable references
- Other operations

Table 11.2 identifies and describes all of the operators for UDM command expressions.

Operator	Description
EQ	Compares the value on the left to the value on the right. If the two are equal, the result is 1, otherwise it is 0. Both the left and right values can be strings or numbers. If they are strings, the comparison is case insensitive.
NE	Works like the equal operator, except that it results in 1 if the left and right value are not equal and 0 if they are equal.
LT	Results in a value of 1 if the left value is less than the right value, otherwise it results in 0. This is a numeric operator.
GT	Results in a value of 1 if the left value is greater than the right value, otherwise it results in 0. This is a numeric operator.
LE	Results in a value of 1 if the left value is less than or equal to the right value, otherwise it results in 0. This is a numeric operator.
GE	Results in a value of 1 if the left value is greater than or equal to the right value, otherwise it results in 0. This is a numeric operator.
AND	Results in a value of 1 if both the left value and right value are not 0, otherwise it results in 0.
OR	Results in a value of 1 if either the left value or the right value are not 0, otherwise it results in 0.
XOR	Results in a value of 1 if either the left or the right values are not 0, but not both. If both the left and right values are 0, then the result of the XOR operator is 0.
NOT	Unlike all of the operators, the NOT operator has only one operand that appears to the right of the NOT operator. This operation evaluates to one if the operand is zero and zero if the operand is non-zero.
+	Result is the sum of the left and right values.
-	Result is subtracting the right value from the left value.
*	Result is the product of the left and right values.
/	Result is the left value divided by the right value. Assuming integer-only math, the remainder is discarded.
%	Result is the remainder of the left value divided by the right value.

Table 11.2 UDM Command Expressions - Operators

11.3.4 Strings in Expressions

You can use strings as operators in **EQ** and **NE** expressions. However, to avoid ambiguity, strings must be quoted explicitly (even if they are contained in a variable reference); otherwise, an error is generated.

The following expression correctly compares strings:

```
<"yes" EQ "yes">
```

Conversely, the following expression results in an error because the strings being compared are not quoted:

```
<yes EQ yes>
```

For these next two examples, assume that a variable called **myvar** has been defined with a value of **yes**.

This expression results in an error because when **myvar** is referenced, its value (**yes**) is not quoted:

```
<$(myvar) EQ "yes">
```

Instead, the correct expression should be:

```
<"$(myvar)" EQ "yes">
```

Index Position and Sequence

When working with strings, the following policy regarding the inclusion/exclusion of positional length/index and sequences should be followed:

- Positional Index or Length = Include the value at that position in the operations.
- Sequences = Exclude the value in operations.

(See [deletestring](#), [insertstring](#), and [substring](#) in the Universal Data Mover 3.2.0 Reference Guide.)

11.3.5 Examples of Expressions

```
> set x=<2 + 4>
> set y=<$(x) * 10>
> echo "x * y = " <$(x) * $(y)>
x * y = 360
```

```
> set x=4
> set y=2
> set z=<$(x) + $(y) * 10>
> echo "z = $(z)"
z = 60
```

```
> set x=10 z=20
> set q=<$(x) LT $(z)>
> echo "q = $(q)"
q = 1
```

11.4 Script Files

You can execute UDM commands interactively or in batch, depending on the operating system.

On all platforms other than OS/400, the batch method reads the UDM commands from a script file specified by the [SCRIPT_FILE](#) option.

z/OS

[SCRIPT_FILE](#) is not specified, the commands are read from the file allocated to the **UNVSCR** DD statement in the execution JCL.

OS/400

The script file and member can be specified by position as the first two command parameters or by using the **SRCFILE** and **SRCMBR** command parameters.

The UDM interactive method also can read UDM commands from a script file.

Syntactically, there is no differences in the command structure between the two methods, with the exception that commands in the batch method script files can contain parameters.

11.4.1 Invoking UDM in Batch Mode with Commands from a Script File

To launch UDM in batch from the Windows or UNIX command line, use the [SCRIPT_FILE](#) option to specify the filename of the script.

For example:

```
udm -s script_filename
```

Scripts also can have parameters, which are specified in the same name=value format of command parameters. To specify options parameters for a script file on the command line, use the [SCRIPT_OPTIONS](#) option.

For example:

```
udm -s copyfiles.udm -o "file=* source=c:\source dest=c:\destination"
```

In this example, UDM is invoked with a script file name **copyfiles.udm**. It has three parameters that are passed to the script file:

1. **file**, with a value of *****
2. **source**, with a value of **c:\source**
3. **dest**, with a value of **c:\destination**

When UDM has finished executing the **copyfiles.udm**, it will terminate.

The following example shows the same options for a batch execution in z/OS:

```
//jobname JOB CLASS=A,MSGCLASS=X
//STEP1 EXEC UDMPRC
//UNVSCR1 DD *
udm commands??
/*
//SYSIN DD *
-s UNVSCR1 -o "file=* source=c:\source dest=c:\destination"
/*
```

For OS/400 examples, see Sections [Running UDM from a Script](#) and [Running UDM in Batch Mode](#).

11.4.2 Invoking UDM Interactively with Commands from a Script File

You also can invoke scripts directly from the UDM prompt in interactive mode (or as part of a script file itself) using the **call** command. Any parameters to the **call** command are passed on to the script being called.

The following example illustrates the **call** command executing the **copyfiles.udm** script (as identified in the previous example):

```
UDM
UDM>call copyfiles.udm file=* source=c:\source dest=c:\destination
UDM>
UDM>quit
```

Unlike executing a script from the command line, UDM will not exit automatically when it finishes processing a script invoked using the **call** command.



Stoneman's Tip

If you are passing a large number of parameters to a script, you may want to break up the **call** command into multiple lines.

You can do this by putting a **+** at the end of each line break, except for the last line.

However, this method cannot be used for invoking UDM script files with **-s** and **-o** command line options (SCRFIL and OPTIO command parameters under OS/400).

11.4.3 Invoking Scripts from within Scripts

As mentioned in the previous section, scripts can use the `call` command to invoke other scripts. Scripts can be nested up to ten levels deep. As each script finishes processing, control is returned to the script that invoked it immediately following the point of invocation.

When nesting scripts, parameters from higher-level invocations are available to scripts invoked at a lower level. If the same parameter name is used in more than one invocation of a series of nested scripts, the value for the instance of the parameter at the lowest level is used.

11.4.4 Parameter Processing

A parameter is referenced inside of a UDM script using the following format:

`$(parameter_name)`

When a parameter reference is encountered, it is replaced with the value of parameter matching the enclosed name. Continuing with `copyfiles.udm` script example (used previously in this section, [11.4 Script Files](#)), a reference to `$(source)` would be replaced with `c:\source`.

An example of how the `copyfiles.udm` script might look is as follows:

```
cd src=$(source)
cd dst=$(dest)
copy src=$(file)
```

In this example:

1. Transfer server with the logical name `src` would change its directory to `c:\source`.
2. Transfer server with the logical name `dst` would change its directory to `c:\destination`.
3. All files in the `c:\source` directory then would be copied from the first transfer server over to the second.

11.5 Subroutines

UDM's scripting language provides support for subroutines.

Subroutines are portions of the script code that can be called, by name, at any point. This provides a convenient way to reuse common script code.

11.5.1 Usage

There are two parts to a subroutine:

1. Definition Names the subroutine and defines the script code that becomes associated with that subroutine name.
2. Invocation Carries out the work of lines of script associated with a subroutine.

Defining a Subroutine

```
subroutine name  
[script line 1]  
...  
[script line n]  
endsub
```

Invoking a Subroutine

```
callsub name
```

Sequence of Defining / Invoking a Subroutine

A subroutine must be physically defined *before* the `callsub` to the routine is used.

For example, the following subroutine will function correctly:

```
subroutine test
    echo "This is subroutine test"
    echo "$(_halton)"
endsub

echo "This is main()"
callsub test
```

However, this subroutine will fail:

```
echo "This is main()"
callsub test

subroutine test
    echo "This is subroutine test"
    echo "$(_halton)"
endsub
```

Nesting / Recursion of Subroutines

UDM allows subroutine nesting (one subroutine calls another subroutine) and recursion (a subroutine calls itself).

For example, the following illustrates subroutine nesting:

```
subroutine a
    echo "Beginning subroutine A"
    callsub b
    echo "Ending subroutine A"
endsub

subroutine b
    echo "Beginning subroutine B"
    echo "Ending subroutine B"
endsub

callsub a
```


11.5.2 Example

```
subroutine loop_increment
    echo "inside loop_increment: $(LOOP)"
    set LOOP=<$(LOOP) + 1>
endsub

echo "Starting Loop:"
set LOOP=0
if <$(LOOP) LT 1>
    callsub loop_increment
end
if <$(LOOP) LT 2>
    callsub loop_increment
end
if <$(LOOP) EQ 2>
    callsub loop_increment
end
echo "Final Value of LOOP: $(LOOP)"
```

Output

```
Starting Loop:
inside loop_increment: 0
inside loop_increment: 1
inside loop_increment: 2
Final Value of LOOP: 3
```

11.6 UDM Variables

Variables are integral data storage objects in the Universal Scripting Engine.

This section provides information on:

- Variable types
- Variable scope
- Variable reference
- Variable attributes

11.6.1 Variable Types

There are two types of variables:

1. Script
2. Global (user-defined and built-in)

Variable Names

There are no restrictions on variable names except:

- They cannot contain double-quote marks (") or spaces.
- UDM reserves variable names beginning with an underscore (_) for its own internal (built-in) variables (see [11.6.8 Built-in Variables](#)). You cannot create a script variable or user-defined global variable that begins with an underscore (_).

11.6.2 Variable Reference

To obtain the value of a variable, you must create a reference for that variable. The reference can appear anywhere in a script line. It is replaced with the value of the referenced variable.

Referencing the value of a global variable in a script is done exactly the same way as for a script variable:

`$(variable_name)`

For example:

```
set srcfile=myfile.txt
set dstfile=yourfile.txt
copy src=$(srcfile) dst=$(dstfile)
```

11.6.3 Script Variables

Script variables are user-defined variables visible only to a called script, and any of its children, for which they have been defined. When the called script has ended, the script variables' definitions are removed from the scripting engine environment.

You must use the **call** command to define script variables. The variables are specified as parameters in a **call** command, after the script name, that loads and executes a script. They are created during execution of the script.

The value of a script variable cannot be changed once it has been created.

(For detailed information on defining script variables, see Section [11.4 Script Files](#).)

11.6.4 Global Variables

Global variables are variables that are visible at all script levels.

There are two types of global variables:

1. User-defined (see Section [11.6.6 User-Defined Variables](#))
2. Built-in (see Section [11.6.8 Built-in Variables](#))

Global variables are permanent in scope and, once defined, last until the UDM Manager is terminated. Once defined, a global variable cannot be undefined, but its value can be changed by issuing another **set** command.

Issuing the **set** command by itself with no arguments displays all user-defined and built-in variables.

Each global variable includes the following information:

1. Name: identifies a variable as either a UDM pre-defined (built-in) variable (see [11.6.8 Built-in Variables](#)) or a user-defined variable. A variable is referenced in a script by this name.
2. Attribute: optional entry that can be included in a variable reference. It provides additional information about a variable (see [11.6.7 Variable Attributes](#)).
3. Value: assigned by the user and/or provided by UDM, depending on the variable. Built-in variables have pre-defined and/or user-defined values. All user-defined variables have user-defined values.

11.6.5 Scope of Script and Global Variables

A variable's scope is its visibility throughout the scripting environment.

Script variables supersede global variables in precedence. That is, if a script variable has the same name as an existing global variable, any references to that variable name will result in the script variable value, not the global variable value.

If more than one script variable exists with the same name, the script variable defined in the last `call` command has precedence.

For example, if UDM encounters the `$(variable_name)` sequence, it first checks to see if a variable with a matching name was passed into the script it currently is executing. If so, UDM uses this variable's value. If not, UDM goes up the chain of calling scripts and uses the value of the first instance of a variable that it finds that matches the name in the sequence.

If no variable with a matching name was passed into any script along the chain, UDM looks to see if a global variable exists with the name. If one is found, its value is used. If no instances are found anywhere, an error is issued.

Variable Scope Scripts

The following three scripts demonstrate variable scope:

script1.udm

```
set var1="a global variable"
set var2="a global variable"
set var3="a global variable"
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
call script2.udm var1="passed into script2"
                var2="passed into script2"
```

script2.udm

```
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
call script3.udm var1="passed into script3"
```

script3.udm

```
print msg="The value of var1 is $(var1)"
print msg="The value of var2 is $(var2)"
print msg="The value of var3 is $(var3)"
```

Running UDM and calling **script1.udm** produces the following results:

```
Processing script: script1.udm
The value of var1 is a global variable
The value of var2 is a global variable
The value of var3 is a global variable
Processing script: script2.udm
The value of var1 is passed into script2
The value of var2 is passed into script2
The value of var3 is a global variable
Processing script: script3.udm
The value of var1 is passed into script3
The value of var2 is passed into script2
The value of var3 is a global variable
Finished processing script: script3.udm
Finished processing script: script2.udm
Finished processing script: script1.udm
```

11.6.6 User-Defined Variables

User-defined variables are defined using the **set** command. They can have any name — except that they cannot begin with an underscore (**_**) character — and any value.

A user-defined variable can be called within any script or in an interactive session:

```
set variable_name=variable_value
```

The following example creates a user-defined variable called **test**:

```
set test="This is a test."
```

Multiple user-defined variables can be set with a single call to the **set** command, listing each variable's name / value pair in succession, separated by spaces:

```
set varname1=value1 varname2=value2 varname3=value3
```

To assign a value to a variable that has one or more spaces in it, the value must be quoted:

```
set lonvar="This variable has a rather long value."
```

11.6.7 Variable Attributes

In addition to accessing the value of a variable, you can access information about that variable through its attributes.

A variable attribute is referenced by putting a dot (.) after the variable name in a variable reference. For example:

`$(name.attribute)`

There are two variable attributes that can be used for any variable:

1. **`exists`**
2. **`length`**

Note: Some built-in variables have attributes specific to those variables.

exists Attribute

The **`exists`** attribute expands to **`yes`** if a variable with that name exists at any scope; it expands to **`no`** if no variable with that name exists.

Note: Unlike when referencing a variable's value, an error is not issued if the **`exists`** attribute is used for a variable name that does not exist.



Stoneman's Tip

You can use the **`exists`** attribute in combination with the **`if`** statement to determine if a variable exists (and take the appropriate action if it does):

```
if $(filename.exists) EQ yes
  copy src=$(filename)
end
```

length Attribute

The **`length`** attribute expands to the length of the variable's value. If the **`length`** attribute is used for a variable that does not exist, an error is issued.



Stoneman's Tip


You can use the **`length`** attribute in combination with the **`if`** statement to decide if a file name is too long to copy to a remote system:

```
if $(filename.length) LE 8
  copy src=$(filename)
end
```

11.6.8 Built-in Variables

UDM provides built-in variables that are used to make available some of its internal values to UDM commands. Depending on the variable, their values are provided by UDM and/or defined via the **set** command.

Note: All built-in variables are preceded by an underscore (**_**) to indicate that they are built-in variables reserved by UDM.



Version 1.1.0 of UDM had four built-in variables: **echo**, **halton**, **lines**, and **rc**.

The names of these variables were not preceded with an **_** as they are in version 3.1.0 and later.

For the purpose of backward compatibility, these variables can be referenced by their 1.1.0 names as well.

Stoneman's Tip

[Table 11.3](#) lists all of the UDM built-in variables and provides a link to detailed information about them in this section.

Variable	Description	Page
_date	Displays the current date in the format appropriate for the system's locale.	185
_echo	Specification for whether or not a command is echoes prior to processing.	185
_execrc	Holds the value of the process executed by the last exec command issued.	185
_file	Name of the file for the current iteration in a forfiles loop.	186
_halton	Return code value that causes UDM to terminate if it is greater than 0 and is equalled or exceeded by the return code value in the _rc variable.	187
_keepalive	Interval at which keepalive messages are sent form the UDM Manager to transfer servers.	187
_lastmsg	Contains all of the messages written in the transaction log for the last network- or file-oriented command issued.	187
_lastrc	Holds the return code of the last command issued and, optionally, an indication of what happened with the last executed statement.	188
_lines	Specification for whether or not the line number is printed with the error if a command cannot be parsed or is malformed.	189
_path	Absolute path of the file for the current iteration in a forfiles loop.	189
_rc	Current UDM return code.	189
_time	Current time.	190
_uuid	Generates a UUID.	190

Table 11.3 Built-In Variables

`_date`


The `_date` built-in variable displays the current date in the format appropriate for the system's locale.

`_date` has several additional variable attributes:

- `day` resolves to the day of the week.
- `month` resolves to the current month.
- `dd` resolves to a two-digit day of the month.
- `ddd` resolves to the Julian day.
- `mm` resolves to the two-digit month of the year.
- `yy` prints the two-digit year.
- `ww` resolves to the two-digit current week of the year.
- `yyyy` resolves to the four-digit year.

Note: The value of `ww` is zero-based, not one based. That is, the first week of the year is 0, the second week is 1, the third week is 2, and so on.

The `_date` variable can be referenced only in your scripts and cannot be set.



You can use the `_date` variable in combination with the `print` command to display custom date information in UDM's transaction log:

```
print msg="Today is $_date.day), $_date.month) $_date.dd)"
```

Produces the following output:

Today is wednesday, January 19

Stoneman's Tip

`_echo`

The `_echo` built-in variable specifies whether or not a command is echoed prior to processing. It can have a value of either **yes** or **no**:

- If the value is **yes**, each UDM command is echoed prior to processing.
- If the value is **no**, the command is not echoed.

The value of `_echo` can be set using the `set` command, as in the following example:

```
set _echo=yes
```

`_execrc`

The `_execrc` built-in variable holds the value of the process executed by the last `exec` command issued. The return code of the `exec` command itself is stored in `_lastrc` (and `_rc`, if the return code is greater than the current value of `_rc`).

The difference between the two values is that the return code for `exec` (in `_lastrc` and `_rc`) tells you whether `exec` was successful in executing the command that it was supposed to execute, while `_execrc` is the return code of that executed command.

The value of `_execrc` can be set using the `set` command.

`_file`

The `_file` built-in variable contains the name of the file for the current iteration in a `forfiles` loop. `_file` also has special attributes, as shown in [Table 11.4](#), below.

For information on using `_file` and its special attributes, see [_file Variable Attributes](#) in [Section 11.10.1 forfiles Built-In Variables](#).

Attribute Name	Description
accessdate	Date on which the file was last accessed. Format (ISO 8601) is yyyy-mm-dd.
accesstime	Time when the file was last accessed. Format (ISO 8601) is hh:mm:ss.
accesstimestamp	Combination of accessdate and accesstime : yyyy-mm-dd hh:mm:ss. If the file does not have an access time, but does have a access date, 00:00:00 is used for the time portion.
createdate	Date on which the file was created. Format (ISO 8601) format of yyyy-mm-dd.
createtime	Time when the file was created. Format (ISO 8601) is hh:mm:ss.
createtimestamp	Combination of createdate and createtime : yyyy-mm-dd hh:mm:ss. If the file does not have a creation time, but does have a creation date, 00:00:00 is used for the time portion.
moddate	Date on which the file was last modified (referenced for z/OS). Format (ISO 8601) is yyyy-mm-dd.
modtime	Time when the file was last modified (referenced for z/OS). Format (ISO 8601) is hh:mm:ss.
modtimestamp	Combination of moddate and modtime : yyyy-mm-dd hh:mm:ss. If the file does not have a modification time, but does have a modification date, 00:00:00 is used for the time portion.
name	Name of the file (same as referencing <code>_file</code> itself without any attributes).
size	Size of the file (in bytes).
type	Type of file. Values are: <ul style="list-style-type: none"> file directory (also used for PDSs under z/OS) unknown type has meaning in a <code>forfiles</code> statement under OS/400 in the LIB file system: <ul style="list-style-type: none"> If the value of <code>_file.type</code> is directory, the file type is a Physical file. If the value of <code>_file.type</code> is file, the file type is a Save file.

Table 11.4 `_file` Built-in Variable – Special Attributes

`_file` cannot be set using the `set` command.

`_halt`

The `_halt` built-in variable specifies a return code value that causes UDM to terminate if that value is:

- Greater than 0
- Equalled or exceeded by the return code value in the `_rc` variable

Note: If the `_halt` value is 0, and the return code in `_rc` is 0, UDM will not terminate.

Each UDM command has a return code indicating its level of success or failure:

- | | |
|--------------|--------------------------|
| • 0 / none | Success or no error |
| • 4 / warn | Warning has been issued |
| • 8 / error | Error has occurred |
| • 16 / fatal | Fatal error has occurred |

The value of `_halt` can be set using the `set` command. You also can use the convenience values of none, warn, error, and fatal (indicating 0, 4, 8, and 16, respectively) to set the value of `_halt`:

```
set _halt=error
```

`_keepalive`

When a UDM session is established, the UDM Manager periodically sends a keep-alive message to the transfer servers – to which the transfer servers respond – in order to make sure the session is still established.

The `_keepalive` built-in variable contains the interval (in seconds) at which these messages are sent. If it has a value of 0, no keep-alive messages are sent.

You can change this interval by setting the `_keepalive` variable using the `set` command before a session is established:

```
set _keepalive=60
```

`_lastmsg`

The `_lastmsg` built-in variable is a data element (that is, a simple array) that contains all of the messages written in the transaction log for the last network- or file-oriented command that was issued (`open`, `close`, `attrib`, `cd`, `copy`, `copydir`, `delete`, or `rename`).

Whenever a new network- or file-oriented command is issued, the contents of `_lastmsg` is cleared before the command is processed so that `_lastmsg` will contain only messages relating to that command.

If UDM encounters a `print` command while processing a network- or file-oriented command, the value of the `msg` parameter in that command is appended to `_lastmsg` as a new line.

The contents of `_lastmsg` can be listed at any time by issuing the following command:

```
data print=_lastmsg
```

`_lastrc`

The `_lastrc` built-in variable holds the return code of the last command issued.

`_lastrc` also has two special attributes: `message` and `result`.

- `message` contains a human-readable string indicating what happened with the last executed statement.
 - If a command could not be executed or had improper values, the value of `_lastrc.message` is `ERROR`.
 - If a command successfully executed, the value of `_lastrc.message` is either `SUCCESS` or some other message (depending upon the command).
- `result` holds an integer value that indicates the result of the last command executed. The meaning of this value depends on the command. Unless otherwise stated:
 - -1 indicates failure.
 - 0 or a positive value indicates success.

This value does not affect the value of `_rc` and `_lastrc`.

`_lastrc` cannot be set using the `set` command.



Stoneman's Tip

You can use the `_lastrc` variable in combination with the `if` statement to take action based on the return value of the previously issued command:

```
copy src=myfile if $_lastrc EQ 0
  delete src=myfile
end
```

`_lines`

The `_lines` built-in variable specifies whether or not the line number of a command (relative to the script in which it occurred) is printed with the error if the command cannot be parsed or is malformed. It has a value of **yes** or **no**.

- If the value is **yes**, the line number is printed.
- If the value is **no**, the line number is not printed.

`_lines` can be set using the `set` command:

```
set _lines=yes
```

`_path`

The `_path` built-in variable contains the absolute path of the file for the current iteration in a `forfiles` loop (see Section [11.10 forfiles Statement](#)).

`-path` cannot be set using the `set` command.

`_rc`

The `_rc` built-in variable holds the current UDM return code, a numeric value that indicates the highest return code received from processing all UDM commands up to that point. The value of `_rc` is the return code that the UDM Manager returns when it exits.

As with the `_halt` variable, `_rc` can be set, via the `set` command, to either of the following integers or convenience values:

- | | |
|--------------------|--------------------------|
| • 0 / none | Success or no error |
| • 4 / warn | Warning has been issued |
| • 8 / error | Error has occurred |
| • 16 / fatal error | Fatal error has occurred |

For example:

```
set _rc=warn
```

`_time`

The `_time` built-in variable displays the current time.

It has several variable attributes:

- `hh` resolves to the two-digit hour (24-hour time).
- `mm` resolves to the two-digit minute.
- `ss` the number of seconds that have elapsed since the current minute.
- `hs` resolves to the number of hundredths of a second that have elapsed since the last second.

You only can reference `_time` in your scripts; it cannot be set using the `set` command.



Stoneman's Tip

You can use the `_time` variable in combination with the `print` command to display custom time information in UDM's transaction log:

```
print msg="It is now $_time.hh):($_time.mm)"
```

Produces the following output:

Today is now 23:31

`_uuid`

The `_uuid` built-in variable, when referenced, generates a UUID.

For example:

```
echo $_uuid  
1732fd12-7b07-4791-a28a-4cf0776db4f7
```

11.6.9 Logical Name Built-In Variables

When a session is established, built-in variables are created for each transfer server and contain information about each server.

The names of these variables are based on the logical name of the transfer server, preceded by an underscore. If the primary transfer server is not specified (implying a two-party transfer session), its built-in variable will have the name `_local`.

These logical name built-in variables persist only for the duration of the session.

They have three attributes:

1. **host**: contains the host name of the transfer server.
2. **port**: holds the port used to connect to the transfer server over.
3. **user**: contains the userid used to sign into the transfer server.

Examples

The following shows how these built-in variables can be used:

```
open remote=mymachine port=10000 user=me pwd=mypwd
if $_lastrc EQ 0
    print msg="Connected to $_remote.host:$_remote.port"
    print msg="    as $_remote.user from $_local"
end
```

This example produces the following output:

```
Connected to mymachine:10000
    as me from (local)
```

11.7 if Statement

The **if** statement is used to add conditional branching of UDM commands.

An **if** statement consists of:

- Comparison operation.
- Series of UDM commands that are carried out if the comparison operation evaluates to true.
- **end** statement that indicates the end of the **if** statement.

For example:

```
if comparison  
    ...  
    UDM commands  
    ...  
end
```

If the comparison does not evaluate to true, UDM will pick up execution from the line after the **end** statement.

Note: The indentation of commands underneath the conditionals is not required in UDM. This is done for the sake of readability; you can indent lines in your own scripts if and as you see fit.

11.7.1 Comparison Operations

In an **if** statement, a comparison consists of three parts:

1. Left-hand value
2. Comparator
3. Right-hand value

The left-hand and right-hand values can be either:

- Variable reference
- Variable attribute
- Constant

Comparators

A comparator determines the type of comparison to be made between the left-hand and right-hand values.

There are six comparators: EQ, NE, LT, GT, LE GE.

EQ - Equal

The equal comparator, EQ, evaluates to true if both the left and right-hand values are equal to each other. If one or more of the values contains alpha characters (non-numeric), the comparison is case insensitive. That is, a word that is all lower case would be equal to the same word if it were all upper case (for example: **dog** would be equal to **DOG**).

Here are some examples of some if statements using the equal comparator:

```
if $(filename) EQ myfile.txt
    print msg="The name of the file is myfile.txt"
end
```

```
if 8 EQ $_lastrc
    print msg="The last command resulted in an error"
end
```

```
if $(filename.exists) EQ yes
    print msg="The filename variable exists"
end
```

NE - Not Equal

The not-equal comparator, NE, evaluates to true if the left-hand value is not the same as the right-hand value. As with the equal comparator, alpha character comparisons are case insensitive.

The following are some examples of the not-equal comparator:

```
if "C:\Program Files\Universal" NE $(mydir)
    print msg="This is not the Stonebranch application directory"
end
```

```
if 8 NE 0
    print msg="This will always print as 8 is not equal to 0"
end
```

```
if $(filename.exists) NE no
    print msg="The filename variable exists"
end
```

LT - Less Than

The less than comparator, LT, evaluates to true if the left-hand value is less than the right-hand value. The less than comparator performs a numeric comparison.

The following are examples of the less than comparator:

```
if 0 LT 8
    print msg="0 is less than 8"
end

if $_rc LT 8
    print msg="No errors have occurred"
end

if $(filename.length) LT 8
    print msg="The length of the filename is less than 8"
end
```

GT - Greater Than

The greater than comparator, GT, evaluates to true if the left-hand value is greater than the right-hand value. As with the less than comparator, the comparison is between to numeric values as in this example:

```
if 8 GT 0
    print msg="8 is greater than 0"
end
```

LE - Less Than or Equal

The less than or equal comparator, LE, is similar to the less than comparator, except that it evaluates to true if the left-hand value is less than or equal to the right-hand value as in these examples:

```
if 8 LE 8
    print msg="8 is less than or equal to 8"
end

if $(filename.length) LE 8
    print msg="The length of the filename is less than or equal to 8"
end
```

GE - Greater Than or Equal

The greater than or equal comparator, GE, is similar to the greater than comparator, except that it evaluates to true if the left-hand value is greater than or equal to the right-hand value as in this example:

```
if $(filename.length) GE 9
  print msg="The filename is longer than 8 characters"
end
```

11.7.2 Adding an Alternate Path with **else** Statement

Alternate Path without **else** Statement

Often there are occasions where you may want to take one branch if some condition is true and another branch if that condition is false, instead of merely picking up execution after the **end** statement. (Those lines would be executed if the condition was true as well, only after executing the statements inside the **if-end** pair.)

This could be accomplished by two well-phrased **if** statements, one following the other, as in this example:

```
if <$(_rc) GE 8>
    echo "There has been an error"
end
if <$(_rc) LT 8>
    echo "There has not been an error"
end
```

Flaws in this Methodology

However, while this is a perfectly valid method, it suffers from two potential flaws:

First and foremost, people may find such logic difficult to read, thus making your UDM scripts more difficult to maintain, especially by those who had not written them in the first place.

Second, if the comparison operation contains a variable and evaluates to true for the first comparison, it is possible something occurs in the statements inside the **if-end** pair that changes the value of the variable and makes the second comparison evaluate to true as well.

For example:

```
if <$(_lastrc) GE 8>
    echo "The last command was not successful"
end
if <$(_lastrc) LT 8>
    echo "The last command was successful"
end
```

In this example, if the command executed before the first **if** statement resulted in an error, the output would have been as follows:

```
Last command was not successful
Last command was successful
```

This is because the `_lastrc` variable holds the value of the last command executed by UDM. In the example given, the command executed before the first **if** statement resulted in an error (for example, result code = 8) and would result in the first **if** statement evaluating to true.

However, the successful execution of the `print` command inside the first `if` statement would result in `_lastrc` being set to 0, which would in turn mean the second `if` statement would evaluate to true, thus printing the second message. This would not have been what was intended.

In this contrived example, it is rather easy to see what went wrong and come up with a workaround: in this case, creating a new global variable into which to save the value of `_lastrc` - for example: `set newvar=${_lastrc}` - and using the new variable in the comparison operations instead of `_lastrc` as its value would not be overwritten. For longer and more complex scripts, however, this may not be the case.

Alternate Path with `else` Statement

UDM offers an easy solution with the `else` statement. As part of the `if` statement, the `else` statement can be used to provide an alternative path to take if the comparison evaluates to false.

The general format of an `if` statement when an `else` statement is used with it is:

```
if expression
...
[else
...]
end
```

In this `if` statement, the parameter for the statement is an expression. If the expression evaluates to a value that is not equal to zero, the positive branch is taken; otherwise the negative (`else`) branch is taken if one exists.

Examples

```
if <$_rc) EQ 0>
    echo "Everything worked okay"
else
    echo "Something went wrong"
end

if ("$(myvar.exists)" EQ "yes">
    echo "The variable, myvar, has been defined."
end
```

Note: The previous style of UDM `if` statements, shown in the following example, still is valid:

```
if <$_lastrc) GE 8>
    print msg="The last command was not successful"
else
    print msg="The last command was successful"
end
```

11.7.3 Nested Conditionals

For complex and powerful operations, `if` statements can be nested inside of each other.

For example:

```
copy src=$(filename)

if $_lastrc EQ 0
    delete src=$(filename)

    if $_lastrc NE 0
        print msg="The source file could not be deleted."
    end

else
    print msg="The copy operation failed."
end

print msg="The operation completed with a return code of $_rc."
```



Stoneman's Tip

Indenting lines underneath conditionals by putting spaces at the front of them, although not necessary, provides a visual cue that those lines are to be executed due to the evaluation of a conditional.

Using this technique with nested conditionals provides an easy way to tell at which 'level' each of the commands belong.

In addition, leaving a blank line before and after a conditional (not required by UDM) provides a way to visually indicate a block of related script commands.

This improves the readability and maintainability of scripts in the future.

11.7.4 Returning Early Using the **return** Command

At times, it is useful to be able to exit from processing a single script file in the middle of that script file if certain processing conditions are not correct. For this, UDM provides the **return** command, which takes the following format:

```
return [value]
```

The **return** command stops processing of the current script and returns control to the calling script at the point immediately following the script call (just as if the script had executed completely without calling the **return** command). If there was not a calling script, and UDM is not running interactively, UDM will exit. The **return** command also can be followed by an optional value. If this is the case, UDM's return code (held by the `_rc` built-in variable) is set to this value upon executing the **return** command.



Stoneman's Tip

One common use of the **return** command is to exit from a script if the previous operation failed. (The `_halt` variable can be used for this situation if you want to exit from UDM altogether.)

However, if you only want to exit the current script, you can couple the **return** command with an **if** statement and the `_lastrc` built-in variable:

```
if $_lastrc NE 0
    return $_lastrc
end
```

11.8 while Statement

The `while` statement implements a simple `while` loop.

The syntax of the `while` statement is:

```
while expression
...
end
```

In this case, the loop iterates (executing the commands between the `while` and `end` statements) as long as the expression evaluates to a value that is not zero.

If the expression evaluates to a value of zero, code execution picks up at the point immediately following the end of the `while` loop.

For example:

```
set n=1
while <$(n) LE 10>
    echo $(n)
    set n=<$(n) + 1>
end
```


11.9 fordata Statement

The **fordata** statement iterates through a data element, once for each line. For each iteration, a variable provided by the user is set to hold the contents of the line in the data element corresponding to the current iteration.

The syntax of the **fordata** statement is:

```
fordata variable-name=data-element  
...  
end
```

Example

```
set i=1  
loaddata mydata=mydata.txt  
fordata line=mydata  
    echo "$(i): $(line)"  
  
    compare "$(line)" "exit" case=yes  
  
    if <"$_lastrc.message" EQ "MATCH">  
        echo  
        echo "Data contains an 'exit' command"  
        echo  
    end  
  
    set i=<$(i) + 1>  
end
```

If a data element called **mydata.txt** contained the following contents:

```
cd /  
ls -al  
exit
```

Running this script against the contents of **mydata** would produce the following results:

```
1: cd /  
2: ls -al  
3: exit  
Data contains an 'exit' command
```

11.10 forfiles Statement

UDM provides a powerful iterative loop structure, **forfiles**, that iterates through a series of statements for each file found that matches a file specification.

The syntax of the **forfiles** statement is:

```
forfiles logical_name=file_spec
        [sortby=attribute-name[,ascending | descending]]
...
    UDM commands
...
end
```

logical_name is the logical name of a transfer server.

file_spec is the file specification used to select files for the iteration (see Section [11.10.2 forfiles File Specification](#)).

From the specified transfer server, UDM builds a list of files that match the file specification. UDM then executes all of the commands listed between the **forfiles** statement and the **end** statement, once for each file in the list.

The optional **sortby** parameter specifies the name of a special attribute *attribute-name* of the **_file** built-in variable (see [Table 11.4 _file Built-in Variable – Special Attributes](#)). The list of files that match *file_spec* will be sorted based on the value of *attribute-name*. **ascending** and **descending** specify whether the matching files are listed in ascending or descending order. (If neither is specified, the list is sorted in ascending order.)

Since having a **sortby** attribute in the **forfiles** loop implies that the file attributes will be used, the file attributes will be retrieved regardless of whether or not **fileattrib=yes** is present.

Examples

To obtain a file list, sorted by creation date (earliest to latest):

```
forfiles src=*.txt sortby=createdate
    # Do some stuff
end
```

Top obtain a file list, ordered by file size from largest to smallest:

```
forfiles src=*.exe sortby=size,descending
    # Do some more stuff
end
```

An error would be produced if **sortby** was present without a value or if it referred to an attribute that does not exist for the **_file** variable.

11.10.1 forfiles Built-In Variables

The **forfiles** statement utilizes two built-in variables: **_file** and **_path**. These variables are set by UDM to contain the file name and absolute path of each file in the list for each iteration.

For an example, assume the following:

- Windows machine with logical name **nt**.
- Directory **C:\Example** on the Windows machine.
- Three files – **file1.txt**, **file2.txt**, and **file3.txt** – in the directory.

The following script segment prints the file name and absolute path of each file in the directory:

```
forfiles nt=C:\Example\*
  print msg="Filename: $_file    Abs. Path: $_path"
end
```

Executing this would build a file list containing the files: **file1.txt**, **file2.txt**, and **file3.txt**. Since there are three files in the list that was built, UDM would iterate through the loop three times:

1. During the first iteration through the loop, the **_file** variable would contain **file1.txt** and the **_path** variable would contain **C:\Example\file1.txt**.
2. During the second iteration, the **_file** variable would contain **file2.txt** and the **_path** variable would contain **C:\Example\file2.txt**.
3. During the third and final iteration, the **_file** variable would contain **file3.txt** and the **_path** variable would contain **C:\Example\file3.txt**.

This script segment would result in the following output:

```
Filename: file1.txt    Abs. Path: C:\Example\file1.txt
Filename: file2.txt    Abs. Path: C:\Example\file2.txt
Filename: file3.txt    Abs. Path: C:\Example\file3.txt
```

_file Variable Attributes

The _file variable also has special attributes that further define a file (see [Table 11.4 _file Built-in Variable – Special Attributes](#)).

For efficiency reasons, all of these attributes – other than **name** and **type** – are retrieved only as requested. You can request to retrieve the file attributes by adding **fileattrib=yes** to the end of the **forfiles** call.

For example:

```
forfiles src=*.txt fileattrib=yes
    echo "$(_file) is $_file.size) bytes in size."
end
```

If the information for an attribute cannot be obtained, its value is set to an empty string.

z/OS

z/OS datasets store only **createdate** and **accessdate**. There is no time (**createtime** and **accesstime**) associated with these dates, nor do z/OS datasets store **moddate** or **modtime**.

11.10.2 forfiles File Specification

The file specification portion of the **forfiles** statement, **file_spec**, tells UDM how to build its list of files. It takes the same format as the **copy** command file specification and can contain wildcards.

To list all of the members in a PDS on a z/OS system, you can issue the following commands:

```
forfiles zos=MYHLQ.MYPDS(*)
  print msg=$(_file)
end
```

This would print the name of each member in a PDS called **MYHLQ.MYPDS**.

For Windows, UNIX, and OS/400 systems (as well as the HFS file system under z/OS), you can list all of the files in the current directory with the following UDM commands:

```
forfiles local=*
  print msg=$(_file)
end
```

To find all of the files ending in **.txt** in a particular directory, **mydir** in this example, issue the following **forfiles** statement:

```
forfiles local=mydir/*.txt
  print msg=$(_file)
end
```

A question mark (**?**) can be used as a wildcard for a single character. In the previous example, let's assume **mydir** contains the files: **file**, **file1**, **file2**, and **file3.txt**.

Executing the following:

```
forfiles local=mydir/file?
  print msg=$(_file)
end
```

Will result in the following output:

```
file1
file2
```



Stoneman's Tip

The **forfiles** file specification can contain wildcards for any UDM files system.

Under z/OS, however, the wildcards only can be used to reference a member of a PDS or PDS/E and not a data set name.

11.10.3 Breaking Out Using the **break** Command

The **break** command is a powerful command that can be issued from inside of a **forfiles** loop. It causes UDM to stop iterating through the **forfiles** loop and resume execution at the command immediately following the **end** statement marking the end of the loop.



Stoneman's Tip

One use for the **forfiles** statement is to try and copy a series of files, deleting the source file if the copy operation was successful. The following is a sample that accomplishes this task, exiting from the loop if a file cannot be copied or if, after copying a file, it cannot be deleted:

```
forfiles local=*
  copy local=$_file
  if $_lastrc NE 0
    print msg="Could not copy $_path"
    break
  end
delete local=$_file
  if $_lastrc NE 0
    print msg="Could not delete $_path"
    break
  end
end
```

11.11 Creating In-Stream Data with the **data** Command

The **data** command can be used to define in-stream data elements that can be passed as input for other commands, such as the **exec** command.

The syntax for the **data** command is as follows:

```
data [NAME|print=NAME] [resolve=all|defined|no] [end=ENDSEQUENCE]
[DATA]
end|ENDSEQUENCE
```

11.11.1 Creating an In-Stream Data Element

An in-stream data element has four parts:

1. Name
2. Optional variable resolution method
3. In-stream data itself
4. End-of-data marker or end sequence

The name uniquely identifies the data element and is used to refer to the data element.

The optional variable resolution method tells UDM whether to resolve variables wrapped in the **\$()** sequence when the data element is referred to.

- If the resolution method is **all** (default), all variables are resolved and an error is issued if the variable is not defined in UDM.
- If the resolution method is **defined**, only references to variables defined in UDM are resolved and all other **\$()** references are left as is in the data element.
- If the resolution method is **no**, UDM does not try to resolve any variable references in the data when the data element is used.

The data portion of the data element is the actual data that will be used by the command that is referencing that data element.

Note: The data is used as entered, including any leading spaces or tabs; no trimming is done.

The end-of-data marker or end sequence marks the end of the data. By default, this is simply the word **end**. It must appear separately, on its own line. However, it is possible that **end** is valid instream data and you can change the end sequence with the **end** parameter of the **data** command.

Example

The following example shows how to use the `data` command in conjunction with the `exec` command to look through a series of copied files and display lines with the occurrence of some string under UNIX:

```
open remote=yourmachine user=someguy pwd=somepwd

data mydata resolve=all
  grep "this is my sequence" $( _file)
  exit
end

copy local=*.txt

forfiles remote=*.txt
  exec remote cmd=ksh input=mydata
end

close
```

11.11.2 Printing Data Element Information

Issuing the `data` command by itself prints a list of the names of all the data elements that have been defined.

Note: Data elements persist beyond individual UDM transfer sessions.

Issuing the `data` command with the `print` parameter and the name of a data element will print the data in that element.

Continuing with the previous example, issuing:

```
data print=mydata
```

Will produce the following output:

```
----> Begin 'mydata' <----
  grep "this is my sequence" $( _file)
  exit
----> End 'mydata' <----
```

Chapter 12

UDM Transfer Operations

12.1 Overview

This chapter provides information on Universal Data Mover (UDM) Transfer Operations.

See [Chapter 13 Transfer Operations \(z/OS-Specific\)](#) and [Chapter 14 Transfer Operations \(OS/400-Specific\)](#) for transfer information specific to those operating systems.

12.2 Transfer Sessions

12.2.1 Opening a Transfer Session

UDM transfer operations all occur within the context of a transfer session. This section details how to open a UDM session.

Opening a Two-Party Transfer Session

All sessions are established using the `open` command. At its simplest, the `open` command specifies the primary and secondary servers for the session:

```
open logical1=hostname logical2=hostname
```

In this example, `logical1` and `logical2` are the user-assigned logical names of the primary and secondary servers, respectively. Each of these parameters is set to the host name or IP address of the corresponding server.

For two-party transfer sessions, where the UDM Manager acts as the primary server, `hostname` is not the host address of the local machine (this would initiate a three-party transfer with the primary server running on the local machine). Instead, the host address is either the name `local` or the asterisk (`*`) character:

```
open machine1=* machine2=somentmachine
```

In this example, a two-party transfer session is established between the UDM Manager, acting as the primary server with the logical name `machine1`, and another machine with the host name `somentmachine`, with the logical name `machine2`.

An alternate method of establishing a two-party transfer is simply to give the secondary server as a parameter to the `open` command:

```
open machine2=somentmachine
```

In this example, a two-party transfer session is implied. In such cases, the logical name of the UDM Manager / primary server side of the transfer session always will be `local`.

Opening a Three-Party Transfer Session

A three-party transfer session can be opened using the same syntax as a two-party transfer session. However, both the primary and secondary servers must be specified explicitly, and the host name of the primary server must be a valid IP or host address:

```
open machine1=somemvsmachine machine2=somentmachine
```

In this example, a three-party transfer session is established between a machine with the host name **somemvsmachine**, given the logical name **machine1**, and a machine with the host name **somentmachine**, given the logical name **machine2**.



Stoneman's Tip

It is important to keep in mind that the host name of the secondary transfer server should be specified from the point of view of the primary server, since it will be making the connection to the secondary server.

Depending on your network configuration, the host name for the secondary server might be different from the UDM Manager's perspective than that of the primary server's.

12.2.2 Session Options

The examples given thus far show the simplest versions of the **open** command. Additional options can follow each server name, such as the port on which the Universal Broker is listening, the codepage that the server uses for text translation, authentication information, and references for a file from which these options are read (this file may be encrypted, if desired). At the end of the **open** command are optional parameters that specify the type of encryption and compression used for the data transfer operations.

(See [Chapter 6 UDM Commands](#) in the Universal Data Mover 3.2.0 Reference Guide for detailed information on these parameters).



Stoneman's Tip

Unless otherwise specified, UDM transfers file data using the SSL protocol and the **NULL-MD5** cipher suite.

If you do not want to take the performance hit of SSL, and authentication of the transferred data is not required, you may want **encrypt=NULL-NULL** specified as a session option.

However, the **NULL-NULL** cipher suite must be in the cipher list for all UDM servers involved in the transfer.

12.2.3 Closing a Session

When all transfer operations have concluded, you can close a transfer session by issuing a `close` command. At this point, UDM is ready to initiate another transfer session.

Alternatively, if you want to exit UDM, you can issue a `quit` command, which closes the transfer session and exits the UDM Manager.

12.3 File Systems

12.3.1 File System Overview

Platforms can support one or more file systems or file access methods.

- UNIX and Windows support a single hierarchical file system.
- z/OS support three file systems (or file access methods) under UDM:
 1. DSN (data set name, the default when UDM is running under z/OS)
 2. DD (ddname defined by a JCL DD statement)
 3. HFS (the hierarchical file system supported by USS)

(See Section [12.5 z/OS File System](#) for detailed information on z/OS file systems.)

- OS/400 supports two file systems under UDM:
 1. LIB (the default file system)
 2. HFS (limited to the root and QOpenSys file systems under IFS)

(See Section [14.2 OS/400 I/O](#) for detailed information on OS/400 file systems.)

All transfer operations on a given server will take place in the server's current file system. Both servers in a transfer session do not have to be in the same file system. UDM is capable of reformatting data between different file systems.



Stoneman's Tip

The default file system under z/OS is DSN, even if the UDM Manager is executed from USS (UNIX System Services).

12.3.2 Changing the Current File System

Changing the current file system on a server is a simple matter of executing the [filesys](#) command, which has the following format:

```
filesys logical_name[={dd|dsn|hfs|lib}]
```

In this format, the logical name refers to the logical name of the transfer server to send the [filesys](#) command. An optional file system (for example, z/OS's DD, DSN, or HFS) can be specified after the logical name to change the current file system on that server. Sending a [filesys](#) command with just a logical name returns the current file system of the server.

Note: A [filesys](#) value of **dd** is available only on z/OS manager for two-party transfer.

12.4 UDM Common File System

UDM provides a set of consistent capabilities for a diverse set of file systems on many different operating systems. UDM commands attempt to behave in a consistent and predictable manner regardless of the file system or operating system on which UDM is running. In order to do so, UDM behavior is based on a Common File System (CFS) model.

CFS is biased towards the hierarchical file systems found on UNIX, Windows, or HFS (z/OS or OS/400). CFS terminology and commands then are applied to each of the UDM-supported file systems on different operating systems.

12.4.1 Common File System Terminology

UDM attempts to make consist use of file system terminology so that it can be applied consistently to file systems that are not hierarchical.

[Table 12.1](#), below, lists CFS terminology for hierarchical file systems like UNIX, Windows, and HFS:

CFS Term	Description
path	Name of a file, which may or may not include a directory. A path is either an absolute path or a relative path. Examples: <ul style="list-style-type: none">• /home/homer/phone.txt• phone.txt• ../homer/phone.txt
absolute path	Full path name of a file, starting at the root directory, network point, or drive letter. Examples: <ul style="list-style-type: none">• /home/homer/phone.txt• \\FILESERVER\homer\phone.txt• C:\program files\phone.txt
relative path	Path name of a file that is relative to the current working directory. Examples: <ul style="list-style-type: none">• phone.txt• ./phone.txt• ../phone.txt• myfiles/phone.txt• ../homer/phone.txt

CFS Term	Description
file	Name of a file. All files are located in a directory. The name does not include a directory name. Examples: <ul style="list-style-type: none">• phone.txt• editor.exe
directory	Name of a directory. The name does not include a file. It can be absolute or relative. Examples: <ul style="list-style-type: none">• /home/homer• /• .• ..• C:\program files
current directory	Every program that runs on a hierarchical file system has a current directory, also known as the working directory. For most programs, this is the directory from which it was invoked.

Table 12.1 CFS Terminology for Hierarchical File Systems

12.5 z/OS File System

The z/OS data set file system is a flat file system. There are no concepts of directories. The files are more commonly referred to as data sets.

z/OS data sets supported by UDM fall into two major categories:

- **Sequential**
Sequential data set has a data set organization of Physically Sequential (PS).
- **Partitioned**
Partitioned Data Set (PDS) has a data organization of Partitioned Organization (PO), which also includes system managed Partitioned Data Set Extended (PDSE) organization.

A PDS is treated as a directory in CFS. A PDS contains a set of individual files called members, which is analogous to a directory containing a set of files. A PDS member has a maximum length of 8 characters.

[Table 12.2](#), below, associates CFS terminology with z/OS partitioned and sequential data sets. Fully qualified data set names are enclosed in apostrophes.

CFS	Sequential	Partitioned
path	Data set name. A path includes fully qualified names and relative names. Examples: <ul style="list-style-type: none"> • PHONE.DATA • 'MYUID.PHONE.DATA' 	PDS name and member name in parenthesis or just a member name. A path includes fully qualified names and relative names. Examples: <ul style="list-style-type: none"> • JCL.CNTL(JOBAB) • 'MYUID.JCL.CNTL(JOBAB)' • JOBAB
absolute path	Fully qualified data set name. Example: <ul style="list-style-type: none"> • 'MYUID.PHONE.DATA' 	Fully qualified PDS name and member name in parenthesis. Example: <ul style="list-style-type: none"> • 'MYUID.JCL.CNTL(JOBAB)'
relative path	Data set name without one or more leading qualifiers. The name is relative to the current directory. Examples: <ul style="list-style-type: none"> • PHONE.DATA • DATA 	PDS name without its high-level or mid-level qualifiers and a member name enclosed in parenthesis or just a member name. The name is relative to the current directory. Examples: <ul style="list-style-type: none"> • JCL.CNTL(JOBAB) • CNTL(JOBAB) • JOBAB
file	Same as path (data set name). It may be absolute or relative.	Member name only. Example: <ul style="list-style-type: none"> • JOBAB

CFS	Sequential	Partitioned
directory	N/A	PDS name without the member name. The directory name may be relative or absolute. Examples: <ul style="list-style-type: none"> JCL.CNTL 'MYUID.JCL.CNTL'
current directory	Current leading qualifiers. Note: It may be more than one qualifier long. Examples: <ul style="list-style-type: none"> MYUID MYUID.DATA 	Current leading qualifiers. Note: It may be more than one qualifier long or even the full PDS name. Examples: <ul style="list-style-type: none"> MYUID MYUID.JCL 'MYUID.JCL.CNTL'

Table 12.2 CFS Terminology Associated with z/OS Data Sets

UDM is capable of running as a JES batch job. In a batch environment data sets may be allocated dynamically by UDM or UDM may use data sets pre-allocated with JCL DD statements.

The JCL DD statement allocates the data set and defines its to the batch job environment as a ddname that the program uses. Although ddnames are not a different file system, they do have their own naming conventions and behavior relative to UDM's CFS.

[Table 12.3](#), below, associates CFS terminology with z/OS partitioned and sequential data sets allocated to ddnames.

CFS	Sequential	Partitioned
path	ddname defined with a JCL DD statement. Examples: <ul style="list-style-type: none"> DD1 PHONE 	ddname with a member name enclosed in parenthesis or just the member name. Examples: <ul style="list-style-type: none"> INDD(JOBAB) MYDATA(PHONE) PHONE
absolute path	There is only one type of path and that is absolute. Refer to path above. <ul style="list-style-type: none"> DD1 	ddname and member name enclosed in parenthesis. Refer to path above.
relative path	N/A	Member name only. The path is relative to the current directory.
file	Same as path (the ddname). Example: <ul style="list-style-type: none"> DD1 	Member name only. Example: <ul style="list-style-type: none"> JOBAB
directory	N/A	ddname with which a PDS is allocated.
current directory	N/A	Current ddname with which a PDS is allocated. Examples: <ul style="list-style-type: none"> INDD OUTDD

Table 12.3 CFS Terminology Associated with z/OS ddnames

12.6 OS/400 File Systems

Universal Data Mover for OS/400 supports two types of file systems:

- LIB (library) file system supports the original, native database file system.
- HFS file system supports the root and QOpenSys file systems under IFS.

Although UDM can access other IFS file systems, only root and QOpenSys are certified.

Currently, Stonebranch, Inc.:

- Does not support other IFS file systems.
- Recommends that users do not use other IFS file systems.
- Provides no warranty for use of other IFS file systems.
- Certifies that users assume all risks in using other IFS file systems.

Risks involved in the use of non-supported IFS files systems include, but are not limited to:

- Loss of data
- Corrupted data
- Non-recoverable exceptions

12.6.1 HFS

HFS follows the common file system (CFS). It supports stream files under the root and QOpenSys IFS file systems. Users using UDM to access file systems under IFS, other than root and QOpenSys, do so at their own risk.

12.6.2 LIB

UDM for OS/400 supports the following file types of the LIB (library) file system

- Physical files (source and data)
- Save files

Table 12.4, below, associates CFS terminology with these LIB file types.

CFS	Physical Files (Source and Data)	Save Files
path	An absolute or relative path. Examples: <ul style="list-style-type: none"> • MYLIB/DATA(NAMES) • DATA(NAMES) 	An absolute or relative path. Examples: <ul style="list-style-type: none"> • MYLIB/BACKUP • BACKUP
absolute path	A fully qualified name containing a library, file, and member. Example: <ul style="list-style-type: none"> • MYLIB/DATA(NAMES) 	A fully qualified name containing a library and file. Example: <ul style="list-style-type: none"> • MYLIB/BACKUP
relative path	A name without a library. The name is relative to the current directory (library). Example: <ul style="list-style-type: none"> • DATA(NAMES) 	A name without a library. The name is relative to the current directory (library). Example: <ul style="list-style-type: none"> • BACKUP
file	Same as path. It may be relative or absolute.	Same as path. It may be relative or absolute.
directory	N/A	N/A
current directory	Name of the current library in which you are working. Example: <ul style="list-style-type: none"> • MYLIB 	Name of the current library in which you are working. Example: <ul style="list-style-type: none"> • MYLIB

Table 12.4 CFS Terminology Associated with LIB File Types

12.7 Transfer Modes and Attributes

12.7.1 Setting the Transfer Type

There are two basic types of file transfers:

- Binary
Binary transfers move the data as it is, without any translation.
- Text
Text transfers translate the data from the source server's code page to the destination server's code page as it is transferred from one server to another.

The default transfer type for UDM is binary.

To set the transfer type, use the `mode` command.

- To set up UDM for text transfers, issue the following command:
`mode type=text`
- To set up UDM for binary transfers, issue the following command:
`mode type=binary`

Issuing the `mode` command by itself displays the current transfer mode. The `mode` command also can be used tell UDM to trim trailing spaces at the end of each line (or record, for record-based file systems such as `dd` and `dsn` in z/OS).

12.7.2 Transfer Attributes

While the `mode` command is used to control the settings for transfer operations as a whole, the `attrib` command can be used to set up the handling of transfer operations for each side of the transfer session.

The `attrib` command can set transfer attributes that apply to either the primary or secondary server. It takes the following form:

```
attrib lname[={dd|dsn|hfs}] [attribute 1=value1]...[attribute n=valuen]
```

Where **lname** is the logical name of the server, the attributes are to be applied.

By default, any attributes listed in the `attrib` command are applied to the currently selected files system unless a specific file system is assigned to the logical name. In that case, the attributes are applied to the specified file system.

The remainder of the `attrib` command contains a series of attributes and their values, some of which will be discussed in further detail in the remainder of this section. If the `attrib` command is issued with just a logical name, UDM will list the currently set attributes for the corresponding server.



When you change file systems for a server using the `filesys` command, the currently set attributes are those that were applied to that file system type. In other words, attributes are not carried over from one file system to another.

Stoneman's Tip

12.7.3 End of Line Sequence

Text mode transfers have the concept of a line in UDM. For record-oriented file systems, such as z/OS's DD and DSN, and OS/400's LIB, each line is a single record. However, for UNIX, Windows, and the HFS file system under USS and OS/400, there is no inherent structure imposed by the operating system on file data.

To determine what constitutes a line in the data for these types of files, UDM looks for an end of line sequence on the source side of a transfer. This can be any sequence of characters (including a zero length sequence, in which case the entire file is considered to be a single line). UDM determines when it has read a complete line of data when this sequence is encountered.

In addition to the normal printable character sets on each platform, an end of line sequence also can be:

- `\r` character sequence, to denote a carriage return character.
- `\l` sequence, to denote a line feed.
- `\n` sequence, to indicate a new line character.



When UDM transfers a line of text data from one server to another, it does not transfer the end of line sequence. Instead, UDM transfers all of the data in each line up to the end of line sequence.

Stoneman's Tip

The end of line sequence also is used on the destination side of a text transfer. The end of line sequence set for the destination side of the transfer is appended to the end of each line of data.

UDM also does this for record-oriented file systems as well. By managing the end of line sequence this way, UDM easily can be used to translate end of line characters across platforms (such as a transfer from UNIX to Windows), strip end of line characters from the data completely, or even add a completely new end of line sequence for use by other applications. For most operations, though, the end of line sequence will not need to be changed.

eof Attribute

The end of line sequence is set with the **eof** attribute. The default value for **eof** depends on the platform and file system selected:

- For Windows-based platforms, the default value is `\r\n`.
- For UNIX platforms and the HFS file system under USS, the default value is `\n`.
- For the HFS file system under OS/400, the default is FILE, which makes end of line terminator consistent with file CCSID.
- For record-oriented file systems (z/OS's dd and dsn, and OS/400's LIB), the value for **eof** is not set.

To provide consistent **eof** definitions under the OS/400 HFS file system, specific ASCII and EBCDIC values are defined for the symbolic values.

- As ASCII, `\n` = x0A, `\r` = x0D, `\t` = x09 and `\l` = x0A.
- As EBCDIC, `\n` = x15, `\r` = x0D, `\t` = x05 and `\l` = x25.

By default, the file CCSID determines the type of **eof**, ASCII vs. EBCDIC. The default ASCII **eof** is `\n` and the default EBCDIC **eof** is `\r\n`.

It is important to note the difference between **eof** definitions as just described and **eof** characters when transferred as data. Due to code page translations and Unicode mappings that take place during data transfer, translated values may be surprising.

Please refer to appropriate translation tables or Unicode mapping tables to understand the values used when **eof** and other control characters are transferred as data. UDM provides default definitions and allows user-defined **eof** attribute overrides in order to avoid translation surprises and associated difficulties.

The following example sets an end of line sequence of an exclamation point (!) for a transfer server:

```
attrib mylogicalname eof=!
```


12.7.4 Line Length and Line Operations

Note: The attributes discussed in this subsection apply solely to the destination side of the transfer.

Other attributes can be used to manipulate transferred data as well.

The **line1en** attribute is used to specify the length, in characters, of a line of data that has been transferred. This value is independent of the end of line sequence and, for record-oriented file systems, the transfer type. If **line1en** is set to a value other than zero (its default value), UDM will manipulate the data according to the method specified with the **lineop** attribute.

The **lineop** attribute specifies what happens to each line (or record, from z/OS's dd and dsn file systems) of data coming from the source transfer. If the value for **lineop** is *none*, the line/record is written as is, however if its length from the source is greater than the value of **line1en**, UDM issues an error. If the value of **lineop** is *stream*, the data from the source side of the transfer is treated as a single record and is subdivided when it is written as a series of lines or records (depending on the file system), each **line1en** characters in length. If the value of **lineop** is *trunc*, each record or line from the source is truncated so that it is at most **line1en** characters in length. Finally, if the value of **lineop** is *wrap*, each line or record from the source side of the transfer that is longer than **line1en** characters is wrapped into multiple lines/records so that the maximum length of each line on the destination side is at most **line1en** characters long.



Stoneman's Tip

Binary data that is transferred from a Windows or UNIX platform (including HFS under USS) is looked at by UDM as one large line or record of source data. The same can be said when transferring text data from these platforms if the end of line sequence is zero length for the source server or the end of line sequence does not exist in the source data.

Under z/OS (except for the HFS file system), UDM will set the **line1en** attribute to be the same as the **1rec1** allocation option for new data sets or the LRECL DCB attribute of existing data sets if the value of **line1en** is zero. UDM also will set the **lineop** attribute to a value appropriate for the transfer type and destination allocation attributes if **lineop** has previous not be set.

12.8 Copying Files with UDM

12.8.1 Simple Copy Operation

At its core, UDM is meant to copy files from one system to another. This is done with the `copy` command.

The basic format of the `copy` command is:

```
copy sourceName=filespec [destName[=filespec]]
```

The `copy` command copies the file specified on the server with the logical name corresponding to **sourceName** to the server with the logical name corresponding to **destName**.

If no destination file name is given, the source file name is used (absent the directory name, regardless of whether or not it was explicitly specified as part of the source file specification). Likewise, if only a directory is given for the destination file specification, the source file name is appended to the directory when writing the file.

If a destination file name or complete file name and directory are given for the destination file specification, that information is used in writing the destination file, regardless of the source file name.

Examples

The following example copies the file `test.txt` from a machine with the logical name `src` to a file called `test.txt` on the machine with the logical name `dst`:

```
copy src=test.txt dst
```

The following example copies the `test.txt` file residing in `c:\files` from a machine with the logical name `src` to a file called `test.txt` on the machine with the logical name `dst`:

```
copy src=c:\files\test.txt dst
```

The following example copies `test.txt` from `src` to the root of drive C on `dst`. The destination file is also called `test.txt`:

```
copy src=test.txt dst=c:\
```

The following example copies the file `test.txt` from `src` to a file called `test.bak` in the root of drive C on `dst`:

```
copy src=test.txt dst=c:\test.bak
```

**Stoneman's Tip**

If you want the destination file name to be the same as that of the source file name, you do not have to specify the destination system in the **copy** command.

The destination will be implied based on the logical name of the source (if the source is the primary server, the destination is assumed to be the secondary server, and vice versa).

12.8.2 Move Operation

A UDM move operation, using the **move** command, is similar to a copy operation.

The only difference between using a **move** command and a **copy** command is that after you move a file, it is deleted from the source server from which it was moved.

12.8.3 Copying Multiple Files Using Wildcards

In addition to copying single files, the `copy` command can be used to copy multiple files by using wildcards in the source file specification. When wildcards are used, any file matching the source file specification will be copied.

There are two types of wildcards:

1. Asterisk (*) wildcard matches zero or more characters.
2. Question mark (?) wildcard matches a single character.

Here are some examples of wildcard matching given the following filenames in the source directory:

- `test.txt`
- `test1.txt`
- `test2.txt`
- `test3.txt`
- `test.bin`
- `test1.bin`
- `test2.bin`
- `test3.bin`

A source file specification of `*` will copy all of the files in the directory, as will `*.*`, `test*.*`, and `tes?.*`.

A source file specification of `*.txt` will copy the first three files.

A source file specification of `*.bin` will copy the last three files.

A source file specification of `test?.txt` will copy the files `test1.txt`, `test2.txt`, and `test3.txt`.



Stoneman's Tip

Wildcards can be used only in the source file specification, not the destination file specification.


Under some operating systems, it is possible for `*` and `?` to be valid characters in a filename.

When they appear in the destination portion of a UDM copy operation, they are treated as file characters and not as wild cards.

Also, keep in mind that while UDM can copy all of the files at a single directory level in a hierarchical file system, it will not traverse the directory tree and copy files from directories at a lower level than the current directory or the directory explicitly specified in the source file specification.

Wildcards should appear only in the filename portion of the file specification and not as part of the directory itself.

12.8.4 File Extension Attributes



Stoneman's Tip

File extension attributes only come into play when a destination filename is not specified.

UDM considers a file extension to be the character sequence followed by the last period (.) in the filename, not including a dot character appearing as the first character in the filename.

The character sequence specified by **defaulttext** is appended verbatim.

A dot (.) is not implied at the beginning of this sequence and must be explicitly included if it is desired in the destination filename.

Some file systems support file extensions. In the case where the source filename is used as the destination filename, UDM can either add an extension or truncate a file's extension. If the **truncatext** attribute is set to **yes**, the extension of the source filename is truncated when writing the destination file. If the **defaulttext** attribute is set to any value, that value is appended to the end of the source filename when writing the destination file.

12.8.5 File Creation Options

The **createop** attribute determines how the destination file is created. By default it has a value of **new**, which means that a file with the destination filename (either implicitly or explicitly specified) must not exist for it to be successfully written by UDM. If a file with that name does exist when UDM begins a copy operation, an error is issued.

If the value of the **createop** attribute is **replace**, the destination file is created if the destination file does not already exist. If it does exist, it is overwritten with the transferred data. If the value of **createop** is **append** and the file already exists, the data transferred is appended to the end of the data already in the existing file. If the file does not exist, a new file is created.

12.8.6 File Permission Attribute

Under the UNIX operating system, the **mode** attribute specifies the mode (in UNIX parlance), or file permissions, of a file created by UDM in a **copy** operation. Existing files do not have their modes modified by UDM. They retain the file mode that they had before the **copy** operation was initiated.

Note: The **mode** attribute is not to be confused with the **mode** command, which is used to set the type of file transferred (and trim option).

The **mode** attribute is set using the **attrib** command.

The value of **mode** is either a set of three numbers, or nothing. Each number in the set corresponds to one or more individuals for whom access is granted for the file:

- First number Owner of the file.
- Second number Users in the group assigned to the file.
- Third number Everyone else.

The value of each number is the sum of values representing file permissions:

- 0 No permissions.
- 1 Permission to execute the file.
- 2 Permission to write to the file.
- 4 Permission to read from the file.

Thus, a value of 7 for the first number would provide the file owner with permission to read, write, and execute a file. A value of 6 for the second number would provide users in the group assigned to the file with permission to read from the file and write to the file, but not to execute the file. A value of 0 for the third number would provide everyone else with no permissions for the file.

By default, the mode attribute is not set. The default mode of a newly created file by UDM is dependent upon the user's umask or the mode of the source file in a UDM transfer.

Examples

The following example provides the owner, group, and everyone else permission to read, write, and execute the file:

```
attrib local mode=777 (
```

The following example provides the owner permission to read, write, and execute the file; members of the file's group permission to read and execute the file; and everyone else no permissions.

```
attrib local mode=750
```

The following example provides the owner permission to read and write the file; and no permissions to the file's group and everyone else.

```
attrib local mode=600
```

Defaults

By default, the **mode** attribute is not set.

The default mode of a newly created file by UDM is dependent upon either:

- User's **umask** attribute.
- Mode of the source file in a UDM transfer.

The latter case comes into play if both the source and destination instances of UDM are:

- Version 3.2 or greater.
- Running under some form of the UNIX operating system or its derivatives (such as Linux).

The **umask** attribute is used in specifying the mode if a UDM version prior to 3.2.0 is involved in the transfer. Version 3.2.0 (and greater) versions can be changed to behave this way as well by setting the **mode** attribute to 0 on the destination side of the transfer.

For example:

```
attrib dest_logical_name mode=0
```

12.8.7 Destination umask

Under UNIX platforms and the HFS file system under z/OS USS, the **umask** attribute can be used to define the file's permissions in accordance to UNIX standards (see [Defaults](#) in Section [12.8.6 File Permission Attribute](#), above).

See [Table 7.2 Common File System Attributes](#) in the Universal Data Mover 3.2.0 Reference Guide for a detailed description of **umask**.

12.8.8 Transaction-Oriented Transfers

A transaction oriented transfer is a file transfer where the destination file is written using a temporary filename. Once the file transfer has been completed, the file is renamed to the appropriate destination filename. To turn on transaction-oriented transfers, set the **trans** attribute for the server on the destination side of the transfer to yes.

Note: UDM for OS/400 LIB file system does not support transaction-oriented transfers.

12.8.9 Changing the Current Directory in UDM

The **cd** command is an easy way to change the current directory in the UDM Common File System, discussed earlier in this section.

By default, when the manager is involved in a two-party transfer, the current directory for primary server is the path in which the manager was launched under. Under z/OS, this would be the high-level qualifier for the user id the manager is running as. The secondary server (as well as the primary server in a three-party transfer) has a default path of the authenticated user's home directory for hierarchical file systems and a high level qualifier corresponding to the authenticated user for dd and dsn file systems.

A user can change the current path for a specific server by issuing a **cd** command:

```
cd lname[=current-directory]
```

In this example:

- **lname** is the logical name of the transfer server to change its default path.
- **current-directory** is the new current directory to set.

If the **cd** command is issued with only a logical name, UDM displays the current directory for the corresponding transfer server.

12.9 Auditing Transfer Operations

12.9.1 Logging File Transfer Operations

When the message level in the UDM server's configuration is set to **audit**, the server writes audit messages to the broker log for each file transferred.

The following is an example of the audit messages produced:

```
NV3950A [1110470739] Transferring from: host: 'Enderlyn.local'
(10.0.0.101), user: 'root', file: '/Volumes/Archive/VPC
Images/.DS_Store'
```

```
UNV3951A [1110470739] Transferring to: host: 'Aluminum.local'
(10.0.0.100), user: 'kevin', file: '/Users/kevin/Desktop/tmp/.DS_Store'
```

```
UNV3952A [1110470739] Successfully transferred '/Volumes/Archive/VPC
Images/.DS_Store' on 'Enderlyn.local' to
'/Users/kevin/Desktop/tmp/.DS_Store' on 'Aluminum.local'
```

The first message is written when the transfer server receives a request to initiate a file transfer and contains the host name of the source machine, the IP address of the source machine, the user authenticated with UDM at the source of the transfer, and the name of the source file to be transferred.

The second message is written when the destination transfer server acknowledges the file transfer requested and contains the host name of the destination machine, the IP address of the destination machine, the user authenticated with UDM at the destination of the transfer, and the destination filename that will be used.

The third message produced indicates that the file was transferred successfully. This message contains the source and destination filenames and host names.

The UDM Manager also can produce these messages when it is involved in a two-party transfer session (though much of the information will be redundant with its standard information messages) by setting its message level to audit. The manager's audit messages are written to stderr (sysout under z/OS).

12.9.2 Reporting Transfer Progress

For long transfer operations, it is often useful to see periodic indications of the operation's progress.

You can get this information by turning on progress reporting using the report command:

report progress=yes

This will cause the UDM manager to issue periodic updates regarding the progress of a file being transferred. The interval these updates are given is the same as the keep alive interval.

Progress messages look as follows:

1024000 bytes processed

Chapter 13

Transfer Operations (z/OS-Specific)

13.1 Overview

This chapter provides information on Universal Data Mover (UDM) transfer operations that are specific to the z/OS operating system.

- [z/OS I/O](#)
- [UDM Commands under z/OS](#)
- [Copying Load Modules](#)

13.2 z/OS I/O

This section provides an overview of the z/OS file systems.

13.2.1 Data Sets

There are a variety of data sets on z/OS. The UDM-supported data set organizations and data set attributes are listed below.

Data Set Names

A z/OS data set name is composed of one or more qualifiers separated by periods. A data set has a maximum length of 44 characters.

A qualifier has an maximum length of eight characters. The first character of a qualifier must start with A-Z, @, #, or \$. The remaining characters can be 0-9, A-Z, @, #, \$, or -.

The first qualifier is commonly referred to as the high-level qualifier (HLQ).

An example data set name is `SYS1.CEE.CEEPRC`, where:

- `SYS1` is the high-level qualifier.
- `CEE` is the second qualifier.
- `CEEPRC` is the third and last qualifier.

In some applications contexts, the HLQ can be left off. TSO and ISPF are such applications. UDM also behaves in this manner. A distinction is made between a data set that specifies the HLQ and one that does not specify the HLQ. When the HLQ is specified, it is referred to as a fully qualified data set name and is enclosed in apostrophes.

Data Set Organization

A data set's organization is obtained from the VTOC's format-1 DSCB. If the data set is cataloged, the DSCB is only read for non-VSAM catalog entries.

The following organizations are supported:

- Physical Sequential
- Partitioned Organization
- Partitioned Data Set Extended
- Generation Data Set

The following organizations are not supported:

- Indexed sequential
- Direct
- Unmovable
- VSAM

Any organization not listed is undetermined.

Record Format

The following record formats are supported:

- Fixed length
- Variable length
- Undefined length
- Blocked
- Fixed length standard
- Variable length spanned
- ISO/ANSI control character
- Machine control character

Block Size

There are three different types of block sizes:

1. User specified block size that cannot exceed 32,760.
2. System determined block size that cannot exceed 32,760.
3. Block sizes supported by the Large Block Interface (LBI) that permits sizes up to 2G. LBI is supported by DFP on tape devices only at this time.

13.2.2 Generation Data Group and Generation Data Sets

A Generation Data Group (GDG) is a catalog entry used to maintain a group of Generation Data Sets (GDS).

GDS's are referred to with absolute names or relative names:

- Absolute name has the form of GDG.G0000V00.
- Relative name has the form of GDG(n), where:
 - n = 0 for the current GDS
 - n = -1 for the previous GDS
 - n = +1 for a new GDS

Allocation

Allocation attributes for a GDS are obtained differently depending on whether the data set name is an absolute or relative form.

Absolute Name

Allocation attributes for an absolute name are provided like any other data set through JCL keywords or allocation options.

Relative Name

Allocation attributes for relative names are provided with one of the following methods:

1. By referring to a cataloged data set from which attributes are copied.
 - a. DCB=(dsname)
 - b. LIKE=dsname
 - c. REFDD=ddname
2. By referring to a model Data Set Control Block (DSCB) on the volume on which the GDG is cataloged. This cannot be used for SMS managed data sets.
 - a. DCB=(modeldsname,yourattributes)
3. By using the DATACLAS and LIKE allocation keywords.
4. Through the assignment of a DATACLAS by a data class ACS routine.

13.2.3 Catalogs

There are two types of catalogs:

1. Integrated Catalog Facility (ICF)
2. VSAM Catalogs

Note: IBM has dropped support for VSAM Catalogs as of January 1, 2000; UDM does not support them.

Symbolic Names

A catalog entry can be defined with symbolic names for the volume serial number. UDM does resolve the symbolic names when they are found for the volume serial number.

Catalog Entry Types

A catalog entry is defined as a specific type. UDM only supports the non-VSAM type entry. A catalog entry type can be any one of the following:

- Non-VSAM Data Set
- Generation Data Set
- Cluster
- Alternate Index
- VSAM Path
- Alias
- User Catalog Connector
- Tape Volume Catalog Library
- Tape Volume Catalog Volume

13.2.4 Allocation

Data set allocation is the process of obtaining access to the data set.

If the data set already exists, it resides on a device, such as a tape or, more likely, a disk. In order to allocate an existing data set, the device must be known. A volume serial name or number and a unit name or number represents an I/O device in z/OS.

The unit and volume serial number (`volser`) can be specified explicitly or specified implicitly (with a catalog entry).

Allocation can be performed with JCL or dynamically. Dynamic allocation requires allocation attributes to be specified by the user. UDM dynamic allocation of a data set that has been migrated by HSM (or similar three-party product) will result in the data set being recalled. UDM will wait until the recall is complete and then continue processing.

13.3 UDM Commands under z/OS

This section describes the behavior of UDM commands when working with z/OS data sets and ddnames.

13.3.1 attrib (Attribute) Command

z/OS data sets can be allocated statically with JCL DD statements or dynamically with the z/OS Dynamic Allocation service (aka SVC 99).

[Table 13.1](#), below, lists the dynamic allocation attributes that can be specified via the [attrib](#) (Attribute) command. For complete details on an allocation attribute, refer to the IBM JCL Reference.

Note: When performing a z/OS-to-z/OS copy and the destination file system is DSN, UDM uses the following allocation attributes obtained from the source file (assuming the file system is DD or DSN): blksize, dirblocks, dsorg, lrecl, primspace, secspace, and spaceunit.

If you do not want to use the source files values for these allocation attributes, you can override them by issuing an [attrib](#) command with the attributes that you want to change before the copy operation.

Attribute Name	Description
abnormaldisp	<ul style="list-style-type: none"> Disposition of a data set after the job ends abnormally. Equivalent to the third position sub-parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). Default is DELETE: it can be set with the UDM configuration option ALLOC_ABNORMAL_DISP.
avgrec	<ul style="list-style-type: none"> Indication that the unit of allocation space specified with the spaceunit attribute is records and that the primary space and secondary space values are in units of 1's, K's, or M's. Equivalent to the JCL AVGREC parameter: AVGREC=size. No default or configuration option.
blksize	<ul style="list-style-type: none"> Block size with which the data set is allocated. Equivalent to the JCL BLKSIZE parameter: BLKSIZE=size. Default is 27998: it can be set with the UDM configuration option ALLOC_BLKSIZE.
blkszlim	<ul style="list-style-type: none"> Block size limit when there is not block size specified from any source. Equivalent to the JCL BLKSZLIM parameter: BLKSZLIM=size. No default or UDM configuration option.
dataclas	<ul style="list-style-type: none"> SMS data class name. Equivalent to the JCL DATACLAS parameter: DATACLAS=name. No default: it can be set with the UDM configuration option ALLOC_DATACLAS.

Attribute Name	Description
datasetseq	<ul style="list-style-type: none"> Data set sequence number that specifies the relative position of a tape data set on the volume. Equivalent to the data set sequence sub-parameter of the JCL LABEL parameter: LABEL=(datasetseq,,). No default or UDM configuration option.
ddndcbref	<ul style="list-style-type: none"> DCB reference to a ddname. Equivalent to the ddname sub-parameter of the JCL DCB parameter: DCB=ddname. No default or UDM configuration option.
den	<ul style="list-style-type: none"> Tape density to use. Equivalent to the DEN sub-parameter of the JCL DCB parameter: DCB=DEN=density. No default or UDM configuration option.
dirblocks	<ul style="list-style-type: none"> Number of directory blocks to allocate for a partitioned data set. Equivalent to the third positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (, dirblocks), . . .). Default is 20: it can be set with the UDM configuration option ALLOC_DIR_BLOCKS.
dsndcbref	<ul style="list-style-type: none"> DCB reference to a cataloged data set name. Equivalent to the data set name sub-parameter of the JCL DCB parameter: DCB=dsn. No default or UDM configuration option.
dsntype	<ul style="list-style-type: none"> Type of SMS data set to allocate. Equivalent to the JCL DSNTYPE parameter: DSNTYPE=type. No default or UDM configuration option.
dsorg	<ul style="list-style-type: none"> Data set organization with which the data set is allocated. Equivalent to the JCL DSORG parameter: DSORG=org. Default is PS: it can be set with the UDM configuration option ALLOC_DSORG.
expdt	<ul style="list-style-type: none"> Expiration date of the data set. Equivalent to the JCL EXPDT parameter: EXPDT=date. No default or UDM configuration option.
label	<ul style="list-style-type: none"> Data set label type used for mostly tape data sets. Equivalent to the label sub-parameter of the JCL LABEL parameter: LABEL=(,label,,). No default or configuration option.
like	<ul style="list-style-type: none"> SMS data set name from which to model data set attributes. Equivalent to the JCL LIKE parameter: LIKE=dsname. No default or UDM configuration option.
lrecl	<ul style="list-style-type: none"> Logical record length with which the data set is allocated. Equivalent to the JCL LRECL parameter: LRECL=len. Default is 1024: it can be set with the UDM configuration option ALLOC_LRECL.
mgmtclas	<ul style="list-style-type: none"> SMS management class name. Equivalent to the JCL MGMTCLAS parameter: MGMTCLAS=name. No default: it can be set with the UDM configuration option ALLOC_MGMTCLAS.
normaldisp	<ul style="list-style-type: none"> Disposition of a data set after the job ends. Equivalent to the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). Default is CATLG: it can be set with the UDM configuration option ALLOC_NORMAL_DISP.

Attribute Name	Description
password	<ul style="list-style-type: none"> Password for password protected data sets. No JCL equivalent.
primspace	<ul style="list-style-type: none"> Primary amount of space to allocate for the data set. Equivalent to the first positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (primspace,), . . .). Default is 15: it can be set with the UDM configuration option ALLOC_PRIM_SPACE.
pwdprotect	<ul style="list-style-type: none"> Specification for whether or not the data set is password protected. Equivalent to the PASSWORD or NOPWREAD sub-parameters of the JCL LABEL parameter: LABEL=(,{PASSWORD NOPWREAD},,). Value must be either PASSWORD or NOPWREAD.
recfm	<ul style="list-style-type: none"> Record format with which the data set is allocated. Equivalent to the JCL RECFM parameter: RECFM=fmt. Default is VB: it can be set with the UDM configuration option ALLOC_RECFM.
refdd	<ul style="list-style-type: none"> ddname from which to copy SMS data set attributes. Equivalent to the JCL REFDD parameter: REFDD=ddname. No default or configuration option.
retpd	<ul style="list-style-type: none"> Retention period of the data set. Equivalent to the JCL RETPD parameter: RETPD=date. No default or configuration option.
rlse	<ul style="list-style-type: none"> Specification for whether or not to release unused space when the data set is unallocated. Equivalent to the sub-parameter RLSE of the JCL SPACE parameter: SPACE=(,.,,),RLSE). Default is no. There is no UDM configuration option. Setting the attribute value to yes turns on the attribute.
secspace	<ul style="list-style-type: none"> Secondary amount of space to allocate for the data set. Equivalent to the second positional parameter of the second positional parameter of the JCL SPACE parameter: SPACE=(, (, secspace), . . .). Default is 15: it can be set with the UDM configuration option ALLOC_SEC_SPACE.
spaceunit	<ul style="list-style-type: none"> Allocation unit used to specify the space to allocate for the data set. Equivalent to the first positional parameter of the JCL SPACE parameter: SPACE=(unit, . . .). Default is TRK: it can be set with the UDM configuration option ALLOC_SPACE_UNIT.
status	<ul style="list-style-type: none"> Status of the data set to be allocated. Equivalent to the first positional parameter of the JCL DISP parameter: DISP=(status,normaldisp,abnormaldisp). Default is: OLD for input and output data sets that exist, NEW for output data sets that don't exist. Default input status can be set with UDM configuration option ALLOC_INPUT_STATUS. Default output status can be set with UDM configuration option ALLOC_OUTPUT_STATUS.
storclas	<ul style="list-style-type: none"> SMS storage class name. Equivalent to the JCL STORCLAS parameter: STORCLAS=name. No default: default value can be set with UDM configuration option ALLOC_STORCLAS.

Attribute Name	Description
unit	<ul style="list-style-type: none"> Unit on which the data set is allocated. Equivalent to the JCL UNIT parameter: UNIT=unit. Default is SYSALLDA: it can be set with UDM configuration option ALLOC_UNIT.
unitcnt	<ul style="list-style-type: none"> Number of units to allocate for a multi-volume data set. Equivalent to the unit count sub-parameter JCL UNIT parameter: UNIT=(,unitcnt,). No default or UDM configuration option.
volcnt	<ul style="list-style-type: none"> Number of volumes to allocate for a multi-volume data set. Equivalent to the volume count sub-parameter JCL VOL parameter: VOL=(,,volcnt,). No default or UDM configuration option.
volseq	<ul style="list-style-type: none"> Volume sequence number on which a multi-volume data set starts. Equivalent to the volume sequence number sub-parameter JCL VOL parameter: VOL=(,volseq,,). No default or UDM configuration option.
volser	<ul style="list-style-type: none"> Volume serial number on which the data set is allocated. Equivalent to the SER sub-parameter of the JCL VOL parameter: VOL=SER=volser. No default: default value can be set with UDM configuration option ALLOC_VOLSER.

Table 13.1 attrib Command - Dynamic Allocation Attributes

13.3.2 cd (Change Directory) Command

The **cd** (Change Directory) command moves the current position within a file system. Position means different things depending on the file system.

This section describes the behavior of the **cd** command for each file system.

DSN (data set name) File System

The DSN (data set name) file system has no directories. The **cd** command treats each data set qualifier as a directory in regards to traversing and positioning within the data set name space.

UDM initializes the current directory to a high-level qualifier equal to the user identifier with which UDM executes.

A **cd** value can be enclosed in apostrophes ('). One or more qualifiers enclosed in apostrophes replaces the current directory value.

A **cd** value not enclosed in apostrophes is concatenated to the current directory value separated by a period (.), effectively moving up in the hierarchy.

There are two special directory (qualifier) names:

1. Current directory - represented by a single period (.).
Directory name . makes no change.
2. Previous directory - represented by two periods (..).
Directory name .. moves back one qualifier.

Examples

[Table 13.2](#), below, provides examples of positioning within the data set file system using the **cd** command. The examples assume the following:

- User ID of TOM123
- UDM logical name SRV

Current Directory before cd	cd command	Current Directory after cd
TOM123	cd srv=data	TOM123.DATA
TOM123.DATA	cd srv=appl.jcl	TOM123.DATA.APP1.JCL
TOM123.DATA.APP1.JCL	cd srv=..	TOM123.DATA.APP1
TOM123.DATA.APP1	cd srv='GAM789.DATA'	GAM789.DATA
GAM789.DATA	cd srv=..	GAM789
GAM789	cd srv=..	GAM789

Table 13.2 cd Command in DSN File System

DD (ddname) File System

The DD (ddname) file system, like the DSN file system, has no directories. DD is the simplest form of file system in UDM.

A ddname is defined with a JCL DD statement. All the allocation attributes are specified with on the JCL DD statement.

UDM initializes the current directory to blanks in the DD file system.

A cd value specifies an allocated ddname to use as the current directory.

13.3.3 copy (Copy) Command

The **copy** command copies files between two systems. The source and destination files are specified with a file specification. The file specification syntax depends on the file system being referenced. This section describes the syntax and semantics of the copy command's file specification.

z/OS

The **copy** command also can be used to copy load modules (see Section [13.4 Copying Load Modules](#)).

DSN File System

The semantics of a file specification is determined primarily by whether a sequential or a partitioned file is being referenced. A sequential file is treated as a single entity in regards to reading and writing. A partitioned file is treated as a composite of multiple sequential files each operated on individually.

Sequential Data Sets

A file is considered sequential if it has a data set organization of Physical Sequential (PS).

A file is referenced directly as a fully qualified name enclosed in apostrophes (') or as a relative name composed of one or more qualifiers concatenated to the current working directory value to form a fully qualified name. The qualifiers . and .. , which are used in the **cd** command, do not have any special meaning in a file specification. They most likely will result in an invalid fully-qualified data set name.

Included in the sequential category are generation data sets. A data set is considered a generation data set if it had a generation data group catalog entry and the data set name includes a generation relative number (for example: (0), (+1), (-1)).

[Table 13.3](#), below, provides some examples of **copy** command file specifications for sequential data sets. The examples assume a UDM logical name of SRV.

Current Directory	File Specification	Result
TOM123	copy srv=data	TOM123.DATA
TOM123	copy srv=app1.jcl	TOM123.APP1.JCL
TOM123	copy srv='GAM789.APP2.DATA'	GAM789.APP2.DATA

Table 13.3 copy Command File Specifications for Sequential Data Sets

Note: In the case of a destination file specification, if no destination file is specified and the attribute **usefqm** is set to *no* (default) for the source **dsn** transfer server, only the part of the data set name matching the source mask in the copy operation is used as the destination file name. If the attribute **usefqm** is set to *yes* on the source, the destination data set name is composed of the source current working directory concatenated with the source file name (the fully qualified file name).

Partitioned Data Sets

A file is considered partitioned if it has a data set organization of Partitioned Organization (PO) or a system managed type of Partitioned Data Set Extended (PDSE).

A file is referenced directly as a fully qualified name enclosed in apostrophes (') or as a relative name composed of one or more qualifiers concatenated to the current working directory value to form a fully qualified name. The qualifiers . and . . that are used in the cd command do not have any special meaning in a file specification and will most likely result in an invalid fully-qualified data set name.

A partitioned data set member requires an additional member name as part of the file specification. The member name is enclosed within parenthesis as in APP.PDS(DATA1), where APP.PDS is the partitioned data set name and DATA1 is the member name.

Table 13.4 provides some examples of **copy** command destination file specifications for partitioned data sets. The examples assume that the fully qualified names are PDS's and a UDM logical name of SRV.

Current Directory	copy Command	Result
TOM123	copy local=app1 srv=data	TOM123.DATA(APP1)
TOM123.DATA	copy local=app1	TOM123.DATA(APP1)
TOM123.DATA	copy local=app1 srv='GAM789.APP2.DATA'	GAM789.APP2.DATA(APP1)
TOM123.DATA	copy local=app1 srv=PDS(PR01)	TOM123.DATA.PDS(PR01)

Table 13.4 copy Command Destination File Specifications for Partitioned Data Sets

Table 13.5 provides some examples of **copy** command source file specifications for partitioned data sets. The examples assume that the fully qualified names are PDS's and a UDM logical name of SRV.

Current Directory	copy Command	Result
TOM123	copy srv=data(app1) local=app1.txt	TOM123.DATA(APP1)
TOM123.DATA	copy srv=app1 local=app1.txt	TOM123.DATA(APP1)
TOM123.DATA	copy srv='GAM789.DATA(APP1)' local=app1.txt	GAM789.DATA(APP1)

Table 13.5 copy Command Source File Specifications for Partitioned Data Sets

Note:

- Member names are restricted to ISPF member naming conventions.
- The **createop** attribute values REPLACE and NEW apply to the member names and not to the partitioned data set.

DD File System

The semantics of a file specification is determined primarily by whether the ddname being referenced has a sequential or a partitioned data set allocated.

A ddname allocating a sequential data set is referred to as a sequential ddname, and a ddname allocating a partitioned data set is referred to as a partitioned ddname in the following text for purposes of brevity.

Sequential ddnames

A ddname is considered sequential if it allocates a data set with an organization of Physical Sequential (PS). A ddname reference is always a fully qualified name. A ddname must not be enclosed in apostrophes (').

There are three possible sequential ddname destination file specifications:

1. Name of the ddname defined by a JCL DD statement.
2. Current working directory value which is set to the name of a ddname and no destination file specification.
3. Source file specification if the current working directory value set to blanks and no destination file specification is provided.

There is one possible sequential ddname source file specification:

1. Name of the ddname defined by a JCL DD statement.

Table 13.6 provides some examples of **copy** command destination file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	copy local=app1.txt srv=appdd1	APPDD1
APPDD1	copy local=app1.txt	APPDD1
	copy local=app1	APP1

Table 13.6 copy Command Destination File Specifications for Sequential ddnames

Table 13.7 provides an example of **copy** command source file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	copy srv=appdd1 local=app1.txt	APPDD1

Table 13.7 copy Command Source File Specification for Sequential ddnames

Note: The **createop** attribute values REPLACE and NEW are not applicable to the sequential ddname file system.

Partitioned ddnames

A ddname is considered partitioned if it allocates a data set with an organization of Partition Organization (PO) or a system managed type of Partitioned Data Set Extended (PDSE). A ddname reference is always a fully-qualified name. A ddname must not be enclosed in apostrophes (').

A partitioned data set member requires an additional member name as part of the file specification. The member name is enclosed within parenthesis as in APPDD(DATA1), where APPDD is the ddname and DATA1 is the member name.

There are three possible partitioned ddname destination file specifications:

1. Name of the ddname defined by a JCL DD statement followed by a member name enclosed in parenthesis.
2. Current working directory value set to the name of a ddname defined by a JCL DD statement, and a member name specified as the destination file specification.
3. Current working directory value set to the name of a ddname defined by a JCL DD statement, and a member name specified by the source file specification if no destination file specification is provided.

There are two possible partitioned ddname source file specification:

1. Complete name of the ddname defined by a JCL DD statement followed by a member name enclosed in parenthesis.
2. Current working directory value set to the name of a ddname defined by a JCL DD statement and a member name specified as the source file name.

Table 13.8 provides some examples of **copy** command destination file specifications for partitioned ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	copy local=app1.txt srv=appdd1(data1)	APPDD1(DATA1)
APPDD1	copy local=app1.txt srv=data1	APPDD1(DATA1)
APPDD1	copy local=app1	APPDD1(APP1)

Table 13.8 copy Command Destination File Specifications for Partitioned ddnames

Table 13.9 provides an example of **copy** command source file specifications for sequential ddnames. The examples assume that ddname APPDD1 is defined with a JCL DD statement and a UDM logical name of SRV.

Current Directory	copy Command	Result
	copy srv=appdd1(data1) local=app1.txt	APPDD1(DATA1)
APPDD1	copy srv=data1 local=app1.txt	APPDD1(DATA1)

Table 13.9 copy Command Source File Specifications for Sequential ddnames

Note: The **createop** attribute values REPLACE and NEW are applicable to members of a partitioned ddname.

13.4 Copying Load Modules

UDM for z/OS provides the ability to copy load modules.

Note: Version 3.2.0 or greater of UDM must be used on both the source and destination sides of the transfer operation for a load module to be properly copied and usable.

The syntax for copying load modules is the same as any copy operation involving PDS/Es. However, there are some differences in how the copy operation takes place when the command to copy load module(s) is issued.

On the source side of the transfer, UDM uses **IEBCOPY** to unload the load modules, matching the source file mask from the PDS/E in which they reside into a temporary data set. It is this temporary data set that is transferred to the destination system. As a result, when UDM displays the status messages indicating that it is copying a file, it is the name of the temporary file that is displayed, since that is what is actually being transferred.

A temporary file name is also used on the destination side of the transfer. After the temporary file has been transferred, the load modules are 'unpacked,' using **IEBCOPY**, into a staging PDS/E (also using a temporary data set name). This PDS/E is created using the same attributes as the source PDS/E. From there, **IEBCOPY** is called a final time to move the load modules from the staging PDS/E to the final destination PDS/E. At this point, all of the temporary files are cleaned up.

The two-step process on the destination side of the transfer is used in case the blocking of the final destination data set does not match that of the source PDS/E.

13.4.1 Example

Figure 13.1, below, illustrates an example script of a load module transfer.

```
open src=* dst=dst-zos
attrib dst createop=new
copy src='MYHLQ.UDM.TESTLM(*)' dst='YOURHLQ.TEST.LOAD'
```

Figure 13.1 Load Module Transfer Script - Example

This simple script will copy all of the load modules from a PDS/E on the source system named **MYHLQ.UDM.TESTLM** to a newly created PDS/E on the destination system named **YOURHLQ.TEST.LOAD**.

Figure 13.2, below, illustrates an example of the output that is received when running a script such as that illustrated in Figure 13.1.

```
Data session established using cipher: NULL-MD5
Two-party session established with 2 (component 1208550125)
Transfer mode settings:
type=binary
trim=no
Session options:
Keep Alive Interval: 120
Network Fault Tolerant: yes
src: Packaging up the following files in
    'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000'
src:  LM1
src:  LM2
src:  tmp
src: 'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000' is being transferred in binary
    mode
src: 'YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000' will be used as the
    destination filename
dst: Unpacking from 'YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000'
src: 'MYHLQ.UDMTMP.STC07047.R2EED53.N0000000' transfered successfully in
    0:01:55.
src: 10566891 bytes read    10566891 bytes written
src: Transfer operation complete. 1 file(s) copied in 0:01:55.448.
src: 10566891 bytes transferred (91529.44 bytes per second)
```

Figure 13.2 Load Module Transfer Script - Output

At the beginning of the **copy** operation, the source side indicates that it is packaging the load modules into a temporary data set, **MYHLQ.UDMTMP.STC07047.R2EED53.N0000000**. This is the temporary data set that is transferred using a data set name of **YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000** on the destination side. It is from **YOURHLQ.UDMTMP.JOB07063.RD36420.N0000000** that the load modules are unpacked into the temporary staging data set before being moved into the final destination PDS/E, **YOURHLQ.TEST.LOAD**, which was specified in the **copy** command.

As you can see, some of the output from a copy operation involving load modules may vary from the output when copying other types of data sets. However, the nomenclature of the **copy** command has not changed.

Likewise, attributes such as **CREATEOP**, **DIRBLOCKS**, and others work the same way with load modules as they do with other types of data sets. This includes the caveat that the attribute settings must be compatible with the type of data set(s) involved in the transfer.

13.4.2 Error Reporting

It is possible for the **IEBCOPY** portions of a load module transfer to fail. If this occurs, UDM prints the output from **IEBCOPY** in the transaction log.

13.4.3 Special Attributes

UDM uses heuristics in determining the space attributes for allocating the temporary data sets. The volume that these data sets reside is chosen by the system.

The **TMPVOLSER** attribute lets you set the volume on which the temporary files will be allocated.

- Setting this attribute on the source side specifies the location of the temporary sequential data set that will be transferred.
- Setting this attribute on the destination side specifies the volume for the temporary transfer file as well as the volume used by the temporary staging PDS/E.

The **TMPPRIMSPACE**, **TMPSECSpace**, and **TMPSPACEUNIT** attributes specify the amount of primary space and secondary space used when allocating the temporary files as well as the space unit used.

- When set on the source side, these attributes affect the temporary sequential data set that will be transferred.
- When set on the destination side, these attributes are used in allocating the temporary transfer file and the temporary staging PDS/E.

The **TMPDIRBLOCKS** attribute is used only on the destination side. It specifies the number of directory blocks used by the staging PDS/E.

Note: Although you can override these attributes, it is not recommended.

Chapter 14

Transfer Operations (OS/400-Specific)

14.1 Overview

This section describes information that is specific to OS/400 file transfer operations:

- OS/400 I/O
- Codepage - CCSID mappings
- UDM commands under OS/400

14.2 OS/400 I/O

This section describes the file systems and file types available in UDM for OS/400.

14.2.1 File Systems

UDM for OS/400 supports two types of file systems:

1. LIB (Library) file system
2. HFS (IFS: root and QOpenSys)

The default file system for UDM on the OS/400 is LIB.

14.2.2 HFS (for OS/400) File System

HFS follows the CFS rules in Section [12.4 UDM Common File System](#).

It supports stream files under the root and QOpenSys IFS file systems. Users using UDM to access file systems under IFS, other than root and QOpenSys, do so at their own risk.

HFS also provides enhanced `eo1` handling and `eo1` attribute values for the mixed ASCII and EBCDIC environment. See Section [12.7.3 End of Line Sequence](#) for details.

14.2.3 LIB File System

LIB follows the extensions to CFS outlined in Section 8.5.2 LIB.

File Types

UDM for OS/400 supports three file types of the LIB file system:

1. Data Physical Files
2. Source Physical Files
3. Save Files

The type of file created in a copy command on the destination side is governed by the OS/400-specific FILETYPE attribute (see [Table 14.2 OS/400-Specific LIB File Attributes for Creating New Files](#)).

The default file type created in a copy command is a data physical file.

14.2.4 Data Physical Files Support

UDM for OS/400 supports data physical files with a CCSID and with no DDS (default CCSID of 65535).

If a DDS is attached to a data physical file on the source side, that same DDS is used when doing an OS/400 to OS/400 copy on the destination side unless the DDSFILE, DDSLIB, and/or DDSMBR attributes are overridden on the destination side to indicate a different DDS or no DDS is to be used.

The DDS itself is not copied, so it must reside on the destination side.

There is one exception: if the source side is a file created via FTP, the created file has an associated DDS file. The associated DDS specifies a single field and DDS source identified by the file is deleted following completion of the job under which the file was created. When UDM identifies a file created by FTP, it ignores the DDS and copies the file as though no DDS exists.

When copying any file to a destination data physical file with the DDSFILE, DDSLIB, and DDSMBR attributes set to point to the file, library, and member of an existing DDS, that DDS is attached to the destination file.

In either case, whether from the source or explicitly on the destination side, if a DDS is used on the destination side, the resulting file's CCSID is determined by the DDS or by the job CCSID settings if not provided by the DDS.

If the source file has no DDS, or if the destination attributes specify no DDS (or are overridden to do so to prevent the source attributes used in an OS/400 to OS/400 copy), the destination data physical file is created with a CCSID of 65535 (meaning no translation).

UDM will issue an informational message if you try to transfer a source file that has a DDS in text mode that tells the user corruption is likely. This is because text translation on the field level is governed by the DDS. UDM does not support independent field-level text translation.

Caution about Text Mode Transfer of Files with DDS

In general, files with DDS should be transferred using binary mode only.

There are instances when a user may want to use text mode. However, without an advanced user's thorough understanding of CCSID and code page, unexpected results will occur.

As of Universal Products for OS/400, release 3.2.0, when the correct conditions are met, UDM maps the code page attribute associated with the data stream to a CCSID. This occurs only when data is transferred to a data physical file in text mode with an associated DDS file.

This mapping is used on the LIB file open to obtain translation between the data stream and data in fields with CCSIDs other than 65535. The translation is done by OS/400; UDM is in no way involved with this translation process.

14.2.5 Source Physical Files Support

Source physical files have a common, known DDS. This DDS specifies the following record format:

- First six bytes contain a sequence number
- Next six bytes contain a line modification date
- Remaining number of bytes are text data. This length can be between 1 and 32754 bytes for single-byte character systems.

A single attribute, USESRCSEQ (with values of YES or NO) governs whether or not the sequence number and modification date are included in the source record when transferring a source physical file. How this happens depends on the mode type of the transfer. By default, this value is set to NO, meaning sequence numbers and modification dates are to be stripped.

When writing a source physical file, the USESRCSEQ attribute specifies whether or not source sequence information is expected to be included in the source data. If the value is set to NO, UDM generates sequence number and modification date information. Otherwise, the first 12 bytes of each source record contain that information. This value is sent as a source attribute in OS/400 to OS/400 copies, so unless it is overridden, it automatically will tell the destination side if the sequence numbers are in the data. Allowing this option to be set permits the effective copying of source physical files from non-400 systems that already contain sequence number information.

When creating UDM sequence data, two additional destination side attributes are used.

- SEQSTART specifies the starting sequence number of the first record written and range from 0000.01 to 9999.99. The default is 0001.00.
- SEQINCR indicates how much the sequence number is incremented from record to record. Valid values are 00.01 to 99.99. The default value is 01.00.

SEQSTART and SEQINCR are sent as source attributes, but used on the destination side only when a new file is being created. If UDM is replacing or appending to an existing source physical file, the values of SEQSTART and SEQINCR of the existing destination file are used.

14.2.6 Copying Source Physical Files

Source physical files can be copied in both text and binary mode.

There are three possible copy operations involving source physical files:

1. Like Copies of Source Physical File Data
2. Non-Source Physical to Source Physical Copies
3. Source Physical to Non-Source Physical Copies

Like Copies of Source Physical File Data

OS/400 to OS/400 copy: both the source and destination are source physical files.

In this case, if the sequence and date fields are not stripped from the source, this data is written as is to the destination. If the fields are stripped, the SEQSTART and SEQINCR attributes define how the sequence data is generated (described later in this section) and the current date is placed in the date field.

Non-Source Physical to Source Physical Copies

Non-source physical file is the source and a source physical file is the destination.

In this case, the SECSTART and SEQINCR attributes are used to create the sequence number and the date field is seeding with the current date.

Source Physical to Non-Source Physical Copies

Source records are read and formatted as described above and the destination system writes them out as dictated by its attributes.

14.2.7 Save Files Support

Save (SAVF) files are essentially binary archives of data. They may contain one or more objects inside. These objects can be extracted individually or in their entirety.

UDM for OS/400 supports NEW and REPLACE operations for CREATEOP when a Save file is the destination file type. It does not support append operations on Save files.

SAVF to SAVF Transfers

Copying a Save file to a Save file always should be performed via a binary transfer, regardless of the mode type setting. All of the source data is read in binary and written in binary. This type of transfer succeeds only if the CREATEOP is set to NEW or REPLACE.

If the destination Save file already exists, and the CREATEOP is set to APPEND, UDM issues an error and aborts the transfer.

Non-SAVF to SAVF Transfers

As with all cases when a Save file is involved, a binary transfer should be forced. The source data is written to the destination in the form it is read. Only the values of NEW and REPLACE are supported for CREATEOP on the destination side when the destination file type is a Save file.

Note: The non-Save file must be a file that originally was created as a Save file on an OS/400 system and then stored as a binary file on a non-OS/400 system.

SAVF to Non-SAVF Transfers

Data is read and transferred automatically in binary to the destination machine.

14.2.8 File Specifications

File specifications in the LIB file system consist of up to three components - library, file, and member - that take the following form:

LIBRARY/FILE(MEMBER)

Note: Data physical files and source physical files have members. Save files do not.

14.2.9 Wild Cards

In source file specifications in the LIB file system for the delete command and for the copy command and file specifications for the **forfiles** statement, wildcards can appear in the library, file, and/or member portions. An asterisk (*) represents a match of zero or more characters. A question mark (?) represents a match of exactly one character,

Wildcards only apply to the library, file or member portion of the fully qualified file name in which they appear. For example, in the statement **COPY SRC=ABC/DEF***, the wildcard only applies to the file portion of the name and an error will result because the user did not provide a member name. To copy all of the files that begin with DEF, along with all of their members from library ABC, use the format **COPY SRC=ABC/DEF*(*)**. Likewise, to copy all of the files and their members in libraries that begin with ABC, use the format **COPY SRC=ABC*/*(*)**.

In destination file specifications, wild cards are not allowed.

Examples

COPY SRC=ABC*/DEF*

Copies all files beginning with DEF from all libraries beginning with ABC.

COPY SRC=ABC/DEF(*)

Copies all the members in the physical file DEF in the library ABC.

DELETE SRC=MYLIB/MYFILE?

Deletes all files in the library MYLIB starting with MYFILE and containing one additional character.

FORFILES SRC= */ *(*)

Lists all members in all files in all libraries.

14.3 Codepage - CCSID Mappings

Information that is stored, moved, and displayed on OS/400 has a CCSID (Coded Character Set Identifier) number associated with it. UDM uses these CCSID numbers, where appropriate, when creating data files, transferring data, and storing data.

Each language available on OS/400 has an associated CCSID. The CCSID identifies the mapping of numeric representations associated with each letter or symbol represented by the computer. It also identifies the glyphs required to represent those characters and symbols when displayed. UDM is not concerned about the display aspect of a CCSID or the associated data, only about the mapping between these numeric representations.

Code pages provide one mechanism of mapping (translating) between these numeric representations. Another means of representing these mappings is to use two CCSIDs: one for the data origin and another for the data destination. For example, when writing data to a file on OS/400, the data stream being sent to the file has an associated CCSID and the file itself has an associated CCSID. In this way, the operating system knows how to provide the translation between data to be written to a file and the data that is physically on the disk file.

OS/400

For UDM on OS/400, the data stream CCSID is established via the code page to CCSID mapping file which is controlled by the UDM Manager [CODEPAGE_TO_CCSID_MAP](#) configuration file option and the UDM Server [CODEPAGE_TO_CCSID_MAP](#) configuration file option.

Of course, the translation also works the other way around, when data is read from a disk it is translated from the physical disk back to the data stream. One special CCSID is 65535, which indicates that no translation is to take place.

When transferring data between computer systems UDM allows the specification of a code page for each system. For example:

```
open source=winsys45 user=id1 pwd=mypwd codepage=iso8859-1
destination=os400trex user=id2 pwd=newpwd codepage=IBM037
```

This tells UDM that the two code pages iso8859-1 and IBM037 are to be used for mapping data between the two systems.

Very often, the numeric portion of a code page also is a CCSID to which the code page relates. In this case, the numeric representations represented by the code page are the same as those represented by the CCSID. One example of this common identification is the code page IBM037 and the CCSID 037. This code page and CCSID represent the native numeric representation of data under OS/400.

The default code page for UDM is IBM037. This is the internal code page, as well as the default external code page used for the control session and data session, unless overridden by the configuration file or the CODEPAGE parameter on the [open](#) command (data session only).

14.3.1 CCSID Mapping

In order to get data to and from a file with a given CCSID, a corresponding CCSID matching the data session code page must be used in order to map the data correctly.

The data stream CCSID is mapped from the code page via the code page to CCSID mapping table. By default, internal tables provide this mapping; however, see the UDM Manager [CODEPAGE_TO_CCSID_MAP](#) configuration option or the UDM Server [CODEPAGE_TO_CCSID_MAP](#) configuration option in the UDM Reference Guide regarding setting up an external file.

If a mapping cannot be made, the following occurs:

1. Warning is issued to the user.
2. Copy operation fails.

ASCII code pages can map to CCSIDs that are available in the HFS file system but not the LIB file system. If one of these code pages is used, a different warning should be issued that lets the user know that the mapping will work for HFS, but the behavior in the LIB file system is indeterminate.

[Table 14.1](#), below, contains those mappings.

Note: If a code page contains a dash (-) in the name (for example, ISO8859-1), an underscore (_) must replace the (-) when the code page is used in a UDM script.

Code Page	CCSID	HFS	LIB
IBM037	037	✓	✓
IBM273	273	✓	✓
IBM277	277	✓	✓
IBM278	278	✓	✓
IBM280	280	✓	✓
IBM284	284	✓	✓
IBM500	500	✓	✓
IBM1047			
IBM1140	1140	✓	✓
IBM1141	1141	✓	✓
IBM1142	1142	✓	✓
IBM1143	1143	✓	✓
IBM1144	1144	✓	✓
IBM1145	1145	✓	✓
IBM1146	1146	✓	✓
IBM1147	1147	✓	✓
IBM1148	1148	✓	✓

Code Page	CCSID	HFS	LIB
ISO8859-1	819	✓	
ISO8859-2	912	✓	
ISO8859-4	914	✓	
ISO8859-5	915	✓	
ISO8859-6	1089	✓	
ISO8859-7	813	✓	
ISO8859-8	916	✓	
ISO8859-9	920	✓	
ISO8859-13	921	✓	
ISO8859-15	923	✓	
PC437	437	✓	
PC737	737	✓	
PC775	775	✓	
PC850	850	✓	
PC852	852	✓	
PC855	855	✓	
PC857	857	✓	
PC860	860	✓	
PC861	861	✓	
PC862	862	✓	
PC863	863	✓	
PC864	864	✓	
PC865	865	✓	
PC866	866	✓	
PC869	869	✓	
PC874	874	✓	
WIN1250	1250	✓	
WIN1251	1251	✓	
WIN1252	1252	✓	
WIN1253	1253	✓	
WIN1254	1254	✓	
WIN1255	1255	✓	
WIN1256	1256	✓	
WIN1257	1257	✓	
WIN1258	1258	✓	

Table 14.1 CCSID Mappings

14.4 Command Reference

This section describes UDM command behavior when working with the LIB and HFS file systems.

14.4.1 attrib (Attribute) Command

UDM provides three attribute levels. In order of precedence, from lowest to highest, they are:

1. Default (lowest priority)
2. Source
3. Override (highest priority)

When a user sets an attribute, the override attribute level is being set. Default attributes are those set by UDM at startup. Source attributes are attributes that UDM obtains from the source file and uses for the destination file. For example, when transferring a file one OS/400 LIB location to another, UDM reads the record length of the source file and uses the source file record length to create the destination file. If the source file is on UNIX or in the HFS file system, record length has no meaning and the source attribute is not set.

In addition to the standard UDM file attributes (CREATEOP, EOL, LINELEN, LINEOP, PADLINE, and TRUNCEXT), OS/400-specific file attributes are required in order to create new files in the LIB and HFS file systems. However, not all attributes are required for all file types. (For information on which attributes can be used with each file type, refer to OS/400 online documentation.)

File Attributes

The following file attributes tables ([Table 14.2](#) and [Table 14.3](#)) provides the following information about OS/400-specific file attributes and the file types for which they are required:

Name

Name of the attribute to be used in the LIB file system

Description

Description of the attribute

Source

Indication of whether or not the attribute is a source attribute. A source attribute is one whose destination side value is taken from its source side unless the user explicitly has overridden the destination side value. Source attributes are used only when both of the these conditions apply:

- Copying from one OS/400 system to another
- Both source and destination are in the LIB file systems

Value

Values that can be assigned to the attribute

UDM Default

UDM default value that is assigned to an attribute if a value is not otherwise assigned. A UDM default of NULL identifies the attribute as available to be set, but not set initially. If the attribute value is NULL (or empty string), the system default is used.

System Default

OS/400 system default value that is assigned to an attribute if its UDM default is NULL or empty string. An attribute can have different system defaults for different file types.

File Type

Types of files in the LIB and HFS file systems to which an attribute applies:

- LIB Library
- PF Data physical file
- SP Source physical file
- SAVF Save file
- Stream

LIB File System Attributes

Table 14.2 identifies attributes that are unique to the LIB file system.

Name	Description	Source	Value	UDM Default	System Default	File Type
ACCPH	Access path type		ARRIVAL, KEYED	NULL	ARRIVAL	SP
ALWDLT	Allow delete operation	✓	YES, NO	NULL	YES	PF, SP
ALWUPD	Allow update operation	✓	YES, NO	NULL	YES	PF, SP
ASPDEV	ASP device		ASP, ASPGRPPRI, SYSTEM, device name	NULL	ASP	LIB
ASPNUM	ASP number		LIBASP, 1-32, ASPDEV	NULL	LIBASP for SAVF, 1 for LIB	LIB, SAVF
AUT	Authority		LIBCRTAUT, ALL, CHANGE, EXCLUDE, USE ²	NULL	LIBCRTAUT for LIB,PF,SP EXCLUDE for SAVF	LIB, PF, SP, SAVF
CCSID *	CCSID of the file (source physical files only). For data physical files, the DDS (if one is given) determines its CCSID; if no DDS is given, the value is 65535.	✓	EBCDIC CCSIDs	CODEPAGE		SP

Name	Description	Source	Value	UDM Default	System Default	File Type
CRTAUT	Create authority		SYSVAL, ALL, CHANGE, EXCLUDE, USE, authority name	NULL		LIB
DDSLIB	Library of the DDS used to describe the file	✓		empty string		PF
DDSFIL	File of the DDS used to describe the file	✓		empty string		PF
DDSMBR	Member of the DDS used to describe the file	✓		empty string		PF
DLTPCT	Maximum percentage of deleted records allowed		1-100, NONE	NULL	NONE	PF
EXPDATE	Expiration date for member	✓	date, NONE	NULL	NONE	PF, SP
FILETYPE	Type of file to create when creating a new file	✓	DATA, SRC, SAVF	DATA		PF, SP, SAVF
FRCRATIO	Records to a force write		integer, NONE	NULL	NONE	PF, SP
GENLVL	Generation severity level		0-30	NULL	20	PF
LIBTYPE	Type of library created when creating a library	✓	PROD, TEST	PROD		LIB
LVLCHK	Record format level check	✓	YES, NO	NULL	YES	PF
MAXMBRS	Maximum number of members	✓	integer, NOMAX	NULL	1 for PF, NOMAX for SP	PF, SP
MAXRCDS	Maximum number of records		1-2146762800, NOMAX	NULL		SAVF
OPTION	Source listing options		SRC, NOSRC, SOURCE, NOSOURCE, LIST, NOLIST, SECLVL, NOSECLVL, EVENTF, NOEVENTF (up to four repetitions)	empty string		PF
RCDLEN	Record length if no DDS is used	✓	integer	92		PF, SP
REUSEDLT	Reuse deleted records	✓	YES, NO	NULL	NO	PF
SEQSTART	Beginning sequence number used when writing to a source physical file	✓	0000.01 – 9999.99	1.00		SP
SEQINCR	Amount to increment sequence number by when writing a record to a source physical file	✓	00.01 – 99.99	1.00		SP
SHARE	Share open data path	✓	YES, NO	NULL	NO	PF, SP, SAVF

Name	Description	Source	Value	UDM Default	System Default	File Type
SIZE	Member size	✓	<ul style="list-style-type: none"> Single values: NOMAX Other values: Comma-separated element list Element 1: Initial number of records 1-2147483646, Element 2: Increment number of records Integer, Element 3: Maximum increments Integer (EX: 10000,1000,3) 	size_attr configuration file entry if provided; otherwise empty string	10000,1000,3 for PF, 10000,1000,499 for SP	PF, SP
USESRCSEQ	Sequence number and modification date information: <ul style="list-style-type: none"> On Source side: retain this information when copying a source physical file On Destination side: Record data includes this information 	✓	YES, NO	NO		SP
WAITFILE	Maximum file wait time		integer, IMMED, CLS	NULL	30 for PF IMMED for SP, SAVF	PF, SP, SAVF
WAITRCD	Maximum record wait time		integer, IMMED, NOMAX	NULL	60	PF, SP
<p>* With CCSID set to CODEPAGE, when the UDM CCSID attribute is not set either explicitly or implicitly via an OS/400 to OS/400 file transfer, the CCSID associated with the code page via the code page to CCSID mapping tables gets used as the CCSID attribute value. One implication is that, by default, files may be created with the CCSID associated with the codepage option.</p>						

Table 14.2 OS/400-Specific LIB File Attributes for Creating New Files

HFS Attributes

[Table 14.3](#) identifies attributes that are unique to the HFS file system. (Currently, there is only one HFS unique attribute, CCSID.)

Name	Description	Source	Value	UDM Default	System Default	File Type
CCSID	CCSID of the file	✓	EBCDIC and ASCII CCSIDs	CODEPAGE		stream

Table 14.3 OS/400 -Specific HFS File Attributes for Creating New Files

14.4.2 call (Call) Command

To invoke a script, the member name is required and can be ***FILE**:

```
call mylib/myfile(myscript)
```

Specifying ***FILE** invokes the normal default OS/400 file search order.

To invoke a script included as an inline file in a database job, the call must specify ***FIRST** as the database member name.

The following example illustrates both:

- Invocation of an inline script, **CALLME**, using the **STRUDM** command from a database job.
- Invocation of an inline script, **CALL1**, using the **CALL** command from a database job.

```
//BCHJOB JOB(testcall) ENDSEV(10) OUTQ(mytest/UDMOUTQ) LOGCLPGM(*YES)
LOG(2 20 *SECLVL) MSGQ(*USRPRF)
//DATA FILE(CALL1) ENDCHAR(ENDDATAFILE)
print msg="I made it to call1 - an inline file"
ENDDATAFILE
//DATA FILE(CALLME) ENDCHAR(ENDDATAFILE)
OPEN S=AS400V5 USER=qatest PWD=***** PORT=4311
CALL CALL1(*FIRST)
CLOSE
ENDDATAFILE
STRUDM SCRFILE(CALLME)
//ENDBCHJOB
```

14.4.3 cd (Change Directory) Command

When you authenticate with a UDM Server running under OS/400, the current library is set to the default library for that user.

In file operations where the library is not identified explicitly as a part of the file specification, the current library is used instead.

Example:

```
COPY SRC=C:\MYFILE DST=MYFILE(MYMEMB)
```

With a current library set to MYUSER, this command will result in a destination file specification name of MYUSER/MYFILE(MYMEMB).

You can change the current library by issuing the cd (Change Directory) command with the new library name as in this example:

```
CD DST=YOURUSER
```

There is a special case, when using UDM from one OS/400 machine to another, where the source library name can be used instead (see Section 14 Samples for more details). In order for this to work, you must first clear the destination current library by issuing the following command:

```
CD DST=.
```

14.4.4 copy (Copy) Command

In both the HFS and LIB file systems, if a file with multi-byte characters, including DBCS (Double Byte Character Set), is transferred using UDM in text mode, data loss or corruption can occur. This is because UDM is basically SBCS (Single Byte Character Set) in nature.

If an SBCS code page is used for the data transfer in text mode, some data can be translated into characters that do not translate back to the same data when written to the target file.

To transfer these type of files, users normally should use binary mode and should be very careful if they find it necessary to use text mode.

14.4.5 File Specification Rules

File specifications can appear in a variety of UDM commands, from **copy** to **forfiles**.

On OS/400, a simple set of rules governs how the full file specification used in an operation is constructed.

Since there are still subtle differences between source and destination side file specifications, in terms of how they are derived, separate rules are provided for each type of specification:

- Source File Specification
- Destination File Specification
 - Source and Destination in LIB
 - Destination (only) in LIB

Source File Specification Rules

The following rules apply to file specifications that are in the source position in a **copy** command.

(In all examples, CURLIB is the current library.)

1. If the file specification contains only the file portion, the current library is pre-pended to the name to refer directly to a file with no member component.

Example:

COPY SRC=MYFILE

The absolute path derived would be CURLIB/MYFILE.

2. If the file specification contains only file and member portions, the current library is pre-pended to the name to refer to a specific member in a file.

Example:

COPY SRC=MYFILE (MYMBR)

The absolute path derived would be CURLIB/MYFILE(MYMBR).

3. If the file specification contains only library and file portions, an absolute path without a member component is used.

Example:

COPY SRC=MYLIB/MYFILE

The absolute path would be exactly as given: MYLIB/MYFILE.

4. If a file specification contains library, file, and member portions, all of those components are used explicitly in the absolute path.

Example:

COPY SRC=MYLIB/MYFILE (MYMBR)

The absolute path would be MYLIB/MYFILE(MYMBR).

Destination File Specification Rules

Destination path names follow many of the same rules as source path names, with one big exception: all or part of the destination path name may be derived using a name (or names, in the case of OS/400 to OS/400 LIB file system copies) coming from the source side of a transfer operation.

Source and Destination in LIB File System

The following rules apply for OS/400 to OS/400 transfers where both the source and destination are operating in the LIB file system.

In these example, the current destination library is DSTLIB and the absolute path of the source file being copied is MYLIB/MYFILE(MYMBR).

1. If the destination file specification contains an empty path (no library, file, or member portions), the file and member portions are derived from the source path. If the destination file is to be a save file, the absolute path in this case would be DSTLIB/MYFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/MYFILE(MYMBR).

Examples:

COPY SRC=MYLIB/MYFILE(MYMBR)

The result is a destination name of DSTLIB/MYFILE(MYMBR) if the destination file type is a physical file.

COPY SRC=MYLIB/MYFILE

The result is a destination name of DSTLIB/MYFILE if the destination file type is a save file.

2. If the destination file specification contains only a file portion, the current library is pre-pended to the absolute path. In this case, if the destination file is to be a save file, the absolute path would be DSTLIB/YOURFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/YOURFILE(MYMBR).

Examples:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURFILE

The result is a destination name of DSTLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

COPY SRC=MYLIB/MYFILE DST=YOURFILE

The result is a destination name of DSTLIB/YOURFILE if the destination file type is a save file.

3. If the destination file specification contains only a file portion (with an empty member), the result is exactly the same as when just a destination file name is given.

Example:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURFILE()

The result is a destination name of DSTLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

4. If the destination file specification contains only file and member portions, the resulting absolute path is DSTLIB/YOURFILE(YOURMBR) if a physical file is wanted.

Example:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURFILE(YOURMBR)

The result is a destination name of DSTLIB/YOURFILE(YOURMBR) if the destination file type is a physical file.

5. If the destination file specification contains only a library portion, that library is used instead of the current library. In this case, an absolute path of YOURLIB/MYFILE is used if a save file is wanted. If a physical file is wanted, an absolute path of YOURLIB/MYFILE(MYMBR) is used.

Examples:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/

The result is a destination name of YOURLIB/MYFILE(MYMBR) if the destination file type is a physical file.

COPY SRC=MYLIB/MYFILE DST=YOURLIB/

The result is a destination name of YOURLIB/MYFILE if the destination file type is a save file.

6. If the destination file specification contains only library and file portions, an absolute path of YOURLIB/YOURFILE is derived if a save file is wanted. If a physical file is wanted, YOURLIB/YOURFILE(MYMBR) is used.

Examples:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/YOURFILE

The result is a destination name of YOURLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

COPY SRC=MYLIB/MYFILE DST=YOURLIB/YOURFILE

The result is a destination name of YOURLIB/YOURFILE if the destination file type is a save file.

7. If the destination file specification contains library and file portions, as well as an empty member name, the result is exactly the same as when the file specification contains only library and file portions.

Example:

COPY SRC=MYLIB/MYFILE(MYMBR) DST=YOURLIB/YOURFILE()

The result is a destination name of YOURLIB/YOURFILE(MYMBR) if the destination file type is a physical file.

8. If the destination file specification contains a complete absolute path (library, file, and member portions), the source file name has no effect on the destination path in any way. In this case, if the destination file type is a save file, YOURLIB/YOURFILE is used. If the destination file type is a physical file, YOURLIB/YOURFILE(YOURMBR) is used.

Examples:

COPY SRC=MYLIB/MYFILE (MYMBR) DST=YOURLIB/YOURFILE (YOURMBR)

The result is a destination name of YOURLIB/YOURFILE(YOURMBR) if the destination file type is a physical file.

COPY SRC=MYLIB/MYFILE DST=YOURLIB/YOURFILE

The result is a destination name of YOURLIB/YOURFILE if the destination file type is a save file.

9. In cases where a member is specified explicitly in the destination file name and the destination file type is a save file, an error is issued.

Note: If the user issues a **cd dst-logical-name=.** command to blank out the current library on the destination side, the library name in the absolute path of the source file is used in the destination absolute path in cases where no library is specified explicitly.

This works only for OS/400 to OS/400 copies where both operating systems are operating in the LIB file system.

Example:

CD DST=.

COPY SRC=MYLIB/MYFILE (MYMBR)

The result is a destination of MYLIB/MYFILE(MYMBR), using the source's library, file, and member names, because none are supplied explicitly in the **copy** command. The current directory on the destination side is empty because the command **cd DST=.** was issued.

Destination (only) in LIB File System

Transfers where only the destination is operating in the LIB file system produce slightly different results.

The following rules apply for:

- Copies from non-OS/400 machines to an OS/400 machine operating in the LIB file system
- Copies from OS/400 machines working in the HFS file system to an OS/400 machine operating in the LIB file system,

These operations do not make use of source attributes describing all the library and file portions of the source file specification.

(in the following example, the source file being copied is MYFILE and the current library on the destination side is MYDSTLIB.)

1. If the destination file specification contains an empty path (no library, file, or member portions), the source file name is used for the file and member names on the destination side. If the destination file is to be a save file, the absolute path in this case would be DSTLIB/MYFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/MYFILE(MYFILE).

Example:

COPY SRC=MYFILE

DSTLIB/MYFILE(MYFILE) will be used as the destination name if the destination file type is a physical file and DSTLIB/MYFILE will be used if the destination file type is a save file.

2. If the destination file specification contains only a file portion, the current library is pre-pended to the absolute path and the source file name is used for the member (if it applies). In this case, if the destination file is to be a save file, the absolute path would be DSTLIB/YOURFILE. If the destination file is to be a physical file, the absolute path would be DSTLIB/YOURFILE(MYFILE).

Example:

COPY SRC=MYFILE DST=YOURFILE

DSTLIB/YOURFILE(MYFILE) will be used as the destination name if the destination file type is a physical file and DSTLIB/YOURFILE will be used if the destination file type is a save file.

3. If the destination file specification contains only a file portion and an empty member portion, the result is exactly the same as when the file specification contains only file portion.

Example:

COPY SRC=MYFILE DST=YOURFILE()

DSTLIB/YOURFILE(MYFILE) will be used as the destination name if the destination file type is a physical file.

4. If the destination file specification contains only file and member portions, the resulting absolute path is DSTLIB/YOURFILE(YOURMBR) if a physical file is wanted.

Example:

COPY SRC=MYFILE DST=YOURFILE (YOURMBR)

DSTLIB/YOURFILE(YOURMBR) will be used as the destination name if the destination file type is a physical file.

5. If the destination file specification contains only a library portion, that library is used instead of the current library. In this case, an absolute path of YOURLIB/MYFILE is used if a save file is wanted. If a physical file is wanted, an absolute path of YOURLIB/MYFILE(MYFILE) is used.

Example:

COPY SRC=MYFILE DST=YOURLIB/

YOURLIB/MYFILE(MYFILE) is used as the destination name if the destination file type is a physical file or YOURLIB/MYFILE if the destination file type is a save file.

6. If the destination file specification contains only library and file portions, an absolute path of YOURLIB/YOURFILE is derived if a save file is wanted. If an absolute path of YOURLIB/YOURFILE(MYFILE) is used if a physical file is wanted,

Example:

COPY SRC=MYFILE DST=YOURLIB/YOURFILE

YOURLIB/YOURFILE(MYFILE) is the destination name if a physical file is wanted and YOURLIB/YOURFILE is used if a save file is wanted.

7. If the destination file specification contains library and file portions, as well as an empty member portion, the result is exactly the same as when the specification contains only a library and file portions.

Example:

COPY SRC=MYFILE DST=YOURLIB/YOURFILE

YOURLIB/YOURFILE(MYFILE) is the destination name if a physical file is wanted.

8. If the destination file specification contains a complete absolute path (library, file, and member portions), the source file name has no effect on the destination path in any way. In this case, if the destination file type is a physical file, YOURLIB/YOURFILE(YOURMBR) is used.

Example:

COPY SRC=MYFILE DST=YOURLIB/YOURFILE (YOURMBR)

YOURLIB/YOURFILE(YOURMBR) is the destination if the destination file type is a physical file.

9. In cases where a member is specified explicitly in the destination file name and the destination file type is a save file, an error is issued.

14.4.6 delete (Delete) Command

The delete (Delete) command in the UDM for OS/400 LIB file system takes the following form:

DELETE logical-name=file-mask

delete Command Requirements

The delete command has the following requirements:

- It can be used to remove files and members, but not libraries.

Note: For the protection of the file system, UDM for OS/400 does not allow users to delete libraries.

delete Command Forms

With UDM for OS/400, the file mask, which can contain wild cards in any portion (library, file and member) takes one of the following forms.

Name	Description
DELETE logical-name=LIBRARY/FILE	Deletes any files (including Save files) in the libraries that match the mask.
DELETE logical-name=FILE	Deletes any files (including Save files) in the current directory (library) that match the mask.
DELETE logical-name=LIBRARY/FILE(MEMBER)	Deletes any members where the library, file, and member portions of their fully qualified names match the appropriate elements of the mask
DELETE logical-name=FILE(MEMBER)	Deletes any members in the current directory whose file and member portions of their fully qualified names match the appropriate elements of the mask

Table 14.4 delete Command Forms with UDM under OS/400

14.4.7 rename (Rename) Command

The **rename** (Rename) command in the UDM for OS/400 LIB file system takes the following form.

RENAME *logical-name old-name new-name*

rename Command Requirements

The **rename** command has the following requirements:

- Libraries cannot be renamed.
- A single object level (file or member) can be renamed only with a single call. The name of a file and one of its members cannot be renamed with a single call. All other cases result in a failure.
- Wild cards are not allowed.
- It can be used only at the file and member level; it cannot be used to rename libraries. However, rename can be used to move existing files to existing libraries.
- It cannot be used to move a member from one file to another, since the destination file may not have the same attributes (for example, record length) as the source file. This could result in corrupt (or seemingly corrupt) data.
- It cannot be used to move a file from one library to another because it should not be used to create new libraries.

rename Command Forms

Name	Description
RENAME logical-name LIBRARY/FILE LIBRARY/FILE	Renames the file in the old portion with file name in the new portion. The library in the new name portion of the rename must match the library name in the old name portion.
RENAME logical-name FILE FILE	Renames the file in the current library in the old portion with the new file name in the current directory (library).
RENAME logical-name FILE LIBRARY/FILE	Renames the file in the current directory with the file name in the new portion. The library in the old name portion must be the current library.
RENAME logical-name LIBRARY/FILE FILE	Renames the file in the given library with the name of the file in the new portion, if the library name in the old portion is the same as the current library
RENAME logical-name LIBRARY/FILE(MEMBER) LIBRARY/FILE(MEMBER)	Renames the old member name with the new member name. Both the library and file name portions in the old and new member names must match
RENAME logical-name FILE(MEMBER) FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name. The FILE portion of each name must be the same
RENAME logical-name FILE(MEMBER) LIBRARY/FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name if the library specified in the old name is the same as the current library and both file name portions match
RENAME logical name LIBRARY/FILE(MEMBER) FILE(MEMBER)	Renames the member in the old name with the name of the member in the new name if the library specified in the new name is the same as the current library and both file name portions match

Table 14.5 rename Command Forms

Chapter 15

Remote Execution

15.1 Overview

This chapter provides information on Universal Data Mover (UDM) remote execution.

UDM provide two commands for remote execution:

- [exec Command](#)
- [execsap Command](#)

15.2 exec Command

15.2.1 Executing Remote Commands within UDM

If you have a licensed version of the Universal Command (UCMD) Manager, version 3.1.1 or later, on the same system with the UDM Manager, you can execute system commands on remote machines using the **exec** command.

The **exec** command has the following format:

```
exec logical-name|host-name [cmd|cmdref|stc]=command [user=userid  
pwd=password] [port=port] [codepage=codepage] [file=filename]  
[xfile=filename] [key=key] [option=option] [mergeLog=yes|no]  
[trace=yes|no] [input=data-element] [svropt=server-options]  
[stdout=data-element] [stderr=data-element]
```

The first parameter of the **exec** command is either:

- Logical name (**logical-name**) of a transfer server (valid only if a transfer session has been established)
- Host name (**host-name**) of the machine on which you want to execute the command.

Note: You must have the UCMD Server and Universal Broker installed on the machine on which the command is to be executed.

The second parameter is the command type, which is either:

- **cmd** (command)
- **cmdref** (command reference)
- **stc** (started task)

For any of these three types, the value (*command*) is the remote command to be executed. (See the Universal Command 3.2.0 User Guide for more information about command types.)

UDM must authenticate a user on the remote machine in order to execute a command.

- If a logical name is specified in the first parameter, the **user** and **pwd** values are inherited from the same options specified in the [open](#) command for that logical name. These inherited values can be overridden by specifying them explicitly in the **exec** command.
- If a host name is specified in the first parameter, the **user** and **pwd** values must be specified explicitly in the **exec** command.

The **port** and **codepage** values are inherited from the UDM Manager's configuration file unless overridden explicitly in the call to the **exec** command.

- **port** specifies which port the Universal Broker is listening on for the remote machine.
- **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, **port**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **exec** command.

- If a plain text file is used, use the **file** parameter to specify the name of this file.
- If the file was encrypted with Universal Encrypt, use the **xfile** parameter to specify the name of this file.

If an encryption key other than the Universal Encrypt default was used, specify that key with the **key** parameter.

These parameters, and the format of the file containing these parameters, work exactly like the corresponding option in the **open** command.

The **option** parameter is used to pass options to the UCMD Server (see the **SCRIPT_OPTIONS** option for UCMD Manager in the Universal Command 3.2.0 Reference Guide for more details).

Two streams of data come back from the **remote** command. By default, output from standard out and standard error of the remote command are written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **merge1og** option can be set to yes if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and OS/400; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in the UCMD Manager when UDM invokes it via the **exec** command. Likewise, if **trace** is turned off in the UDM Manager, the UCMD Manager is invoked with tracing turned off. You can override this behavior for the UCMD Manager invocation by setting the **trace** option in the call to the **exec** command.

There are some commands that require input from standard input. To provide this input, you must create a data element with the data command containing the input. Specifying the name of the data element with the **input** parameter will cause the information in the data element to be sent over as standard input to the **remote** command.

The **svropt** parameter can be used to override UCMD Server options.

Note: UDM does not require a space before the server options, as does Universal Command.

The **stdout** and **stderr** parameters specify data elements to contain standard out and standard error, respectively, from the remote command. If the data elements do not exist, they are created. If the data elements do exist, they are overwritten with the output from the remote command. If the value portion refers to an existing non-data element variable or the name of a built-in variable (that is, any variable beginning with an underscore), an error is issued.

The **exec** command output will still be written to UDM stdout (the transaction log) and UDM stderr, where appropriate, even with the presence of the **stdout** and/or **stderr**.

15.2.2 Return Values

When the **exec** command is invoked, the return value from the **exec** command indicates whether or not UDM was able to invoke the remote command. The return value from the **exec** command will be 0 (none) if the remote command was invoked.

Upon successful invocation of the remote command, it might be useful to have the return value of the remote command itself in addition to whether or not the remote command could be executed. The remote command's return value is stored in the built-in variable **_execrc**.

15.2.3 exec Command Examples

The following example uses the **exec** command to execute a simple directory listing on a Windows machine that is part of a transfer session:

```
exec winmachine cmd="dir c:\"
```

The following example uses the **exec** command to invoke a started task:

```
exec mvsmachine stc="mytask,param=$(TASK_PARM) "
```

The following example calls the **exec** command that uses a data element for input to the remote command:

```
# Define the data element  
  
data shellinput  
  echo "Comparing $(_file) with $(_file).old:"  
  diff $(_file) $(_file).old  
  exit  
end  
  
# Rename all existing files on the destination  
  
forfiles dst=*  
  rename dst $(_file) $(_file).old  
end  
  
# Copy the new files over and compare them  
  
forfiles src=*  
  copy src=$(_file)  
  exec dst cmd="sh" input=shellinput  
end
```

15.3 execsap Command

15.3.1 Triggering SAP Events within UDM

If you have a licensed version of the Universal Connector (version 3.1.1 or later) on the same system with the UDM Manager, you can execute SAP events using the **execsap** command.

The **execsap** command has the following format:

```
execsap [host=host-name] | destination type=event|generic  
          [eventid=event-id] [parm=event-parm] [client=client]  
          [user=userid] [pwd=password] [codepage=codepage]  
          [file=filename] [xfile=filename] [key=key] [mergeLog=yes|no]  
          [trace=yes|no]
```

Note: UDM does not support the **execsap** command for OS/400 and Windows.

The first parameter of the **execsap** command is either:

- Host parameter with an SAP destination entry
- Name of a destination in your SAP RFC file

The **type** parameter specifies the type of action being performed. A specified type of **event** requires that an SAP event ID be specified with the **eventid** parameter.

Note: For version 3.2.0, the only valid type is **event**, which triggers an SAP event.

An event parameter can be passed to the SAP event using the **parm** parameter.

The **client** parameter specifies the SAP client.

UDM must authenticate a user SAP in order to execute an SAP action. The user ID and password can be specified with the **user** and **pwd** parameters, respectively.

The codepage is inherited from the UDM Manager's configuration file unless explicitly overridden in the call to the **execsap** command. The **codepage** specifies to which codepage the output of the remote command is translated.

The **user**, **pwd**, and **codepage** parameters can be stored in an external file instead of being specified explicitly in the **execsap** command syntax.

- If a plain text file is used, the **file** parameter specifies the name of this file.
- If the file was encrypted with Universal Encrypt, the **xfile** parameter specifies the name of this file.

If an encryption key was used other than Universal Encrypt's default, that key can be specified with the **key** parameter. This options and the format of the file containing the options work exactly like the corresponding option in the **open** command.

Two streams of data come back from the SAP execution. By default, output from standard out and standard error is written to standard out and standard error by the UDM Manager (SYSPRINT and SYSOUT, respectively, under z/OS). The **merge1og** option can be set to **yes** if you want both output streams written to the UDM transaction log (standard out under UNIX, Windows, and OS/400; SYSPRINT under z/OS).

By default, if the UDM Manager is invoked with tracing turned on, tracing will be turned on in Universal Connector (USAP) when UDM invokes it via the **execsap** command. Likewise, if trace is turned off in the UDM Manager, USAP is invoked with tracing turned off. You can override this behavior for the USAP invocation by setting the **trace** parameter in the call to the **execsap** command.

15.3.2 execsap Command Example

The following is an example of executing an SAP event using the **execsap** command:

```
execsap  sapdest type=event eventId=MYEVENT parm=MYPARM +  
        user=me pwd=mypwd
```

Chapter 16

Return Code Processing

16.1 Overview

Universal Data Mover (UDM) return codes, particularly for batch operations, are used to gauge the degree of success of a job. Each job generates a return code that indicates the status of the job when it ended.

16.1.1 UDM Return Codes

Table 16.1, below, organizes UDM return codes into four categories.

Category	Value	Description
Success	0 / none	All commands completed successfully.
Warning	4 / warn	Non-critical error in operation.
Error	8 / error	UDM command failed to execute because it was: <ul style="list-style-type: none">• Inappropriately issued (for example, a copy command was issued before a transfer session had been established).• Malformed; that is, grammatically correct but either:<ul style="list-style-type: none">• Missing required parameters.• Containing invalid parameters.
Fatal	16 / fatal	Fatal error has occurred; UDM cannot continue and must exit. A fatal error is one that prevents UDM from running: <ul style="list-style-type: none">• Failure to allocate memory.• Failure to initialize portions of the UDM application.• Parser errors (grammatically incorrect scripting language).

Table 16.1 UDM Return Codes

Each return code category has an integer value and a convenient value.

Processed commands return only integers as return code values. The convenient values can be used when setting return codes in the `_rc` and/or `_halt` variables via the `set` command (see Section [16.3.1 Return Codes in set \(Set\) Command](#)).

16.2 Return Codes in UDM Built-In Variables

During processing, UDM keeps track of the return codes from processed commands.

`_lastrc` Variable

The `_lastrc` built-in variable holds the return code of the last command issued. It also has a special attribute, `message`, that contains a human-readable string indicating what happened with the last executed statement.

`_rc` Variable

The `_rc` built-in variable holds the highest-numbered return code that UDM has received (and placed in the `_lastrc` built-in variable) from all processed commands during the current session.

`_rc` also can be set via the `set` command (see [16.3 Setting Return Codes](#)).

`_halton` Variable

The `_halton` built-in variable specifies a return code that, if equaled or exceeded by the return code in `_rc`, causes UDM to exit.

Otherwise, when UDM exits, it returns the highest-numbered return code that it received to the UDM Manager.

`_halton` only can be set via the `set` command (see [16.3 Setting Return Codes](#)).

(For detailed information on these variable, see [Chapter 11 UDM Scripting Language](#).)

16.3 Setting Return Codes

16.3.1 Return Codes in set (Set) Command

You can use the **set** command to manage UDM's return code and UDM's action based on this return code. The **set** command lets you set any of the following return code values (integer or convenient) in both the `_haltton` variable and the `_rc` variable:

- 0 / none
- 4 / warn
- 8 / error
- 16 / fatal

The following example sets the value of `_rc` to 0 and the `_haltton` condition to error:

```
set _rc=0 _haltton=error
```



Issuing the **set** command by itself, with no parameters, will display the values of all of the UDM Manager's internal variables that can be set by the user.

Stoneman's Tip

The **set** command also can be used to set other UDM Manager variables. See Section [6.45 set](#) in the Universal Data Mover 3.2.0 Reference Guide for detailed information on using the **set** command.

Note: You cannot use the **set** command to set the `_lastrc` variable.

Issuing the set Command

1. If the **set** command is issued without any parameters (variables), all of the global variables and their current values are displayed.
2. If the **set** command is issued with variable names but no following equal signs (=), the values to which the variables resolve are displayed.
3. If the **set** command is issued with variable names followed by an equal signs (=) but no values, the values are set to an empty string.

16.3.2 Return Codes in return (Return) Command

You also can use the **return** command to set the return code value (integer only) in the `_rc` variable. (See Section [6.42 return](#) in the Universal Data Mover 3.2.0 Reference Guide for detailed information on using this command.)

Appendix A

Examples

A.1 Overview

This chapter provides operating-specific examples that demonstrate the use of Universal Data Mover:

- [UDM Manager for z/OS Examples](#)
- [UDM Manager for UNIX and Windows Examples](#)
- [UDM Manager for OS/400 Examples](#)

The first example in each section provides a line-by-line explanation of that example.

A.2 UDM Manager for z/OS Examples

This section describes how to use UDM in the z/OS environment.

It provides specific examples for the following topics, which explain how to use UDM in a two-party mode between z/OS and UNIX:

- [Copy a File to an Existing z/OS Sequential Data Set](#)
- [Copy a z/OS Sequential Data Set to a File](#)
- [Copy a Set of Files to an Existing z/OS Partitioned Data Set](#)
- [Copy a File to a New z/OS Sequential Data Set](#)
- [Copy a Set of Files to a New z/OS Partitioned Data Set](#)

For each topic, there is an example (as appropriate) for both the DSN and DD file systems.

Note: These z/OS examples apply equally as well to the Windows operating systems, with appropriate changes for the file system syntactical differences.

A.2.1 Copy a File to an Existing z/OS Sequential Data Set

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

DD file system

```
//S1      EXEC UDMPRC
//APOUT   DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=so19 user=top098 pwd=p100m
4 filesys local=dd
5 cd unix=/opt/app/data
6 mode type=text
7 copy unix=data10.txt local=APOUT
8 quit
/*
```

For this first z/OS example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being printed out prior to processing.
2. Line 2 sets the error condition value on which script processing halts. Any error greater than or equal to warn halts script processing.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **so19**. The host **so19** is given the logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if successfully verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the local file system from the default of DSN to DD. The file system type dictates the syntax and semantics of file specifications, such as in the **copy** command.
5. Line 5 changes the current directory of the UDM server **unix** running on host **so19**.
6. Line 6 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages (for example, ASCII and EBCDIC) and process the end-of-line characters.
7. Line 7 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager ddname **APOUT**. Recall that line 4 sets the local file system type to DD; hence, **APOUT** is referencing a ddname.
8. Line 8 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

DSN file system

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 attrib local createop=replace
7 copy unix=data10.txt local='app.data.daily'
8 quit
/*
```

The DSN file system example is basically the same as the DD file system example, with these changes:

- Removal of the `filesys` command (line 4 in the DD file system example), since the default file system for the z/OS manager is DSN.
- Addition of the line 6, which sets the local attribute `createop`.
The `createop` attribute controls how a file is created. By default, its value is *new*, indicating that only new files are created and existing files are not written over (replaced). In this example, the value is being set to *replace*, which specifies that if the file exists, it should be replaced; otherwise, it is created.

A.2.2 Copy a z/OS Sequential Data Set to a File

These examples copy, in text mode, a sequential data set on z/OS to a remote UNIX system.

Note: A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed. Variable record formats do not normally have trailing spaces, so trimming normally is not required.

DD file system

```
//S1      EXEC UDMPRC
//APOUT DD DSN=APP.DATA.DAILY,DISP=SHR
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 filesys local=dd
5 cd unix=/opt/app/data
6 mode type=text trim=yes
7 copy local=apout unix=data10.txt
8 quit
/*
```

DSN file system

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text trim=yes
6 copy local='app.data.daily' unix=data10.txt
7 quit
/*
```

A.2.3 Copy a Set of Files to an Existing z/OS Partitioned Data Set

These examples copy, in text mode, multiple files with one **copy** command to an already allocated partitioned data set (PDS) on a z/OS system.

The file names used to create the member names in the destination PDS are the source file names.

However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). The period in the file extension is not a valid character in PDS member names, so UDM must be instructed to remove the file extensions before copying them into the PDS.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in PDS **APP.DATA.PDS**:

- **DATA001**
- **DATA002**
- **DATA003**

DD file system

```
//S1      EXEC UDMPRC
//APOUT   DD DSN=APP.DATA.PDS,DISP=SHR
//UNVSCR  DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 filesys local=dd
4 cd unix=/opt/app/data
5 mode type=text
6 attrib local truncext=yes
7 copy unix=*.txt local=apout
8 quit
/*
```

DSN file system

```
//S1      EXEC UDMPRC
//UNVSCR  DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt local='app.data.daily'
7 quit
/*
```

A.2.4 Copy a File to a New z/OS Sequential Data Set

This example copies, in text mode, a file from a remote UNIX system to a sequential data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as they are delivered, create a sequential, variable block record data set with a logical record length of 1024.

The sample below changes the record length to 256 in order to demonstrate how to set dynamic allocation attributes.

A DD file system sample is not provided, since creating a new data set with JCL is the same in UDM as it is in any batch application. There are no UDM specific requirements.

DSN file system

```
//S1      EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local lrecl=256
6 copy data10.txt local='app.data.daily'
7 quit
/*
```

Note: All file names in the UNIX system must be within the eight-character range to be transferred successfully.

Almost all data set allocation attributes can be specified as UDM attributes, providing you with the ability to dynamically allocate any supported data set.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and can result in invalid or unintentional attribute combinations.

A.2.5 Copy a Set of Files to a New z/OS Partitioned Data Set

This example copies, in text mode, a set of files from a remote UNIX system to a partitioned data set on z/OS. The data set does not exist on z/OS; UDM is instructed to create it.

The data set is dynamically allocated based on the local UDM dynamic allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults as they are delivered create a sequential, variable block record data set with a logical record length of 1024.

This example changes the data set organization from sequential (PS) to partitioned (PO) and adjusts the data set's space allocation to space units of cylinders, primary space to 1, secondary space to 2, and directory blocks to 10.

DSN file system

```
//S1 EXEC UDMPRC
//UNVSCR DD *
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local dsorg=po spaceunit=cyl primspace=1 secspace=2 +
6 dirblocks=10 truncext=yes
7 copy unix=*.txt local='app.data.pds'
8 quit
/*
```

Note: Line 5 is continued onto line 6 with the line continuation character (+).

A.3 UDM Manager for UNIX and Windows Examples

This section describes how to use UDM in the Windows and UNIX environments.

It provides specific examples for the following topics, which explain how to use UDM in a two-party mode:

- [Simple File Copy to the Manager](#)
- [Simple File Copy to the Server](#)
- [Copy a Set of Files](#)

Each example illustrates a procedure that occurs under the platform's default file system.

(See Section [A.2 UDM Manager for z/OS Examples](#) for z/OS examples that apply equally as well to the Windows operating systems.)

A.3.1 Simple File Copy to the Manager

This example copies, in text mode, one file to another. This is the simplest form of data transfer.

```
1 set _echo=yes
2 set _halton=warn
3 open unix=so19 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy unix=data10.txt
7 quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being printed out prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.
3. Line 3 opens a session between the local UDM Manager and a remote UDM Server running on host **so19**. The host **so19** is given the a logical name of **unix**. The **open** command also provides user credentials for the UDM Server to verify and, if success verified, specifies the user ID with which the UDM Server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **so19**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager as **data10.txt**.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

A.3.2 Simple File Copy to the Server

This example copies, in text mode, a sequential data set on the UDM Manager machine to a remote UNIX system.

```
1 set _echo=yes
2 set _halton=warn
3 open unix=so19 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy local=c:\data10.txt
7 quit
```

For this UNIX and Windows example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being sent to stdout prior to processing.
2. Line 2 sets error condition value on which script process halts. Any error equal to or greater than 4 halts script processing. A value of 4 effectively means halt on any error or warning.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **so19**. The host **so19** is given the a logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if success verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **so19**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** in the root directory on drive C of the Windows machine to the UNIX Server as **data10.txt**.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

A.3.3 Copy a Set of Files

This example copies, in text mode, multiple files with one copy.

This example assumes that the remote UNIX directory `/opt/app/data` contains the following list of files:

- `data001.txt`
- `data002.txt`
- `data003.txt`
- `data004.pr`
- `data005.pr`

The following files will be created on the destination machine:

- `data001.txt`
- `data002.txt`
- `data003.txt`

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt
7 quit
```

A.4 UDM Manager for OS/400 Examples

This section describes how to use UDM in the OS/400 environment.

It provides specific examples for the following topics, which explain how to use UDM in a two-party mode between OS/400 and UNIX:

- [Copy a File to an Existing OS/400 File](#)
- [Copy an OS/400 Data Physical File to a File](#)
- [Copy a Set of Files to an Existing Data Physical File](#)
- [Copy a File to a New OS/400 Data Physical File](#)
- [Copy a File to a New OS/400 Source Physical File](#)
- [Copy a Set of Files to a New Data Physical File on OS/400](#)
- [Copy Different Types of OS/400 Files using forfiles and \\$_file.type](#)
- [Invoke a Script from a Batch Job](#)

Note: These examples apply equally as well to the Windows operating system, with appropriate changes for the file system syntactical differences.

Each topic provides an example for the LIB file system.

The first topic, [Copy a File to an Existing OS/400 File](#), also provides an example specific to the HFS file system. For other examples similar to those used in the HFS file system, see Section [UDM Manager for UNIX and Windows Examples](#).

A.4.1 Copy a File to an Existing OS/400 File

These examples copy, in text mode, one sequential file to another. This is the simplest form of data transfer.

LIB file system

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

For this first OS/400 example, the following is a line-by-line explanation:

1. Line 1 turns on command echo, which results in each command being sent to stdout prior to processing.
2. Line 2 sets the error condition value on which script processing halts. Any error greater than or equal to **warn** halts script processing.
3. Line 3 opens a session between the local UDM Manager and a remote UDM server running on host **so19**. The host **so19** is given the logical name of **unix**. The **open** command also provides user credentials for the UDM server to verify and, if successfully verified, specifies the user ID with which the UDM server executes.
4. Line 4 changes the current directory of the UDM server **unix** running on host **sol9**.
5. Line 5 changes the transfer mode type from binary (the default) to text. Text mode transfers will translate between code pages (for example, ASCII and EBCDIC) and process the end-of-line characters.
6. Line 6 is the **copy** command that actually moves the data between systems. It copies file **data10.txt** on server **unix** to the local UDM Manager library: MYLIB Data Physical File APPDATA member DAILY.
7. Line 7 executes the **quit** command, which closes all sessions and exits UDM with the highest exit code set.

HFS file system

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4     filesys local=hfs
5 cd unix=/opt/app/data
6 mode type=text
7 attrib local createop=replace
8 copy unix=data10.txt local=/opt/appdata
9 quit
```

This HFS file system example is basically the same as the LIB file system example, with these changes:

- Addition of line 4, which changes the local file system from the default of LIB to HFS. The file system type dictates the syntax and semantics of file specifications, such as in the **copy** command.
- Addition of line 7, which sets the local attribute **createop**.
The **createop** attribute controls how a file is created. By default, its value is *new*, which indicates that only new files are created and existing files are not written over (replaced). In this case, its value is being set to *replace*, specifying that if the file exists, it should be replaced; otherwise, it is created.

A.4.2 Copy an OS/400 Data Physical File to a File

This example copies, in text mode, a Data Physical File on OS/400 to a remote UNIX system.

Note: A text transfer, by default, does not trim spaces from the end of a record. If the data set being copied is a fixed record format, each record is padded with spaces so that the record length equals the logical record length of the data set. If you do not want the trailing spaces copied, they must be trimmed.

LIB file system

```
1 set _echo=yes
2 set _halton=warn
3 open unix=sol9 user=top098 pwd=p100m
4 cd unix=/opt/app/data
5 mode type=text trim=yes
6 copy local=MYLIB/APPDATA(DAILY) unix=data10.txt
7 quit
```


A.4.3 Copy a Set of Files to an Existing Data Physical File

This example copies, in text mode, multiple files with one **copy** command to an already allocated Data Physical File on an OS/400 system.

The file names used to create the member names in the destination Data Physical File are the source file names. However, note that file names on UNIX and Windows file systems often have a file extension as part of their name. A file extension is a suffix separated from the file's base name with a period (for example, BASE.TXT). Member names are limited to 10 characters on the OS/400 system, so UDM must be instructed to remove the file extensions before copying them into the PDS.

The **truncext** attribute is used to instruct UDM to remove file name extensions from the source file prior to using the name as the destination member name.

This example assumes that the remote UNIX directory **/opt/app/data** contains the following list of files:

- **data001.txt**
- **data002.txt**
- **data003.txt**
- **data004.pr**
- **data005.pr**

The result of the copy operation will create the following members in Data Physical File APPDATA:

- **DATA001**
- **DATA002**
- **DATA003**

LIB file system

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local truncext=yes
6 copy unix=*.txt local=MYLIB/APPDATA
7 quit
```

A.4.4 Copy a File to a New OS/400 Data Physical File

This example copies, in text mode, a file from a remote UNIX system to a data physical file on OS/400. The Data Physical File does not exist on OS/400; UDM is instructed to create it.

The file type created defaults to a Data Physical File. The Data Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80, and the maximum members to unlimited (*nomax*), in order to demonstrate how to set allocation attributes.

LIB file system

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local rcdlen=80 maxmbrs=nomax
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and can result in invalid or unintentional attribute combinations.

A.4.5 Copy a File to a New OS/400 Source Physical File

This example copies, in text mode, a file from a remote UNIX system to a Source Physical File on OS/400. The Source Physical File does not exist on OS/400; UDM is instructed to create it.

The Source Physical File is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a Data Physical File with a logical record length of 92 and maximum members of 1.

This example changes the file type to **src** in order to demonstrate how to set allocation attributes.

LIB file system

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local filetype=src
6 copy unix=data10.txt local=MYLIB/APPDATA(DAILY)
7 quit
```

Almost all data set allocation attributes can be specified as UDM attributes giving you the ability to dynamically allocate any supported Data Physical File.

Care should be taken that conflicting allocation attributes are not specified. The results of the allocation should be checked to verify they meet your intentions. Although UDM checks attribute values, some values are provided by the system from sources that UDM cannot verify and may result in invalid or unintentional attribute combinations.

A.4.6 Copy a Set of Files to a New Data Physical File on OS/400

This example copies, in text mode, a set of files from a remote UNIX system to a data physical file on OS/400. The data file does not exist on OS/400; UDM is instructed to create it.

The data set is allocated based on the local UDM allocation attributes. UDM provides default attributes that can be changed to meet local requirements. The UDM defaults, as delivered, create a data physical file with a logical record length of 92 and maximum members of 1.

This example changes the record length to 80 and the maximum members to unlimited (*nomax*).

LIB file system

```
1 set _echo=yes _halton=warn
2 open unix=sol9 user=top098 pwd=p100m
3 cd unix=/opt/app/data
4 mode type=text
5 attrib local maxmbrs=nomax rcdlen=80 truncext=yes
6 copy unix=*.txt local=MYLIB/APPDATA
7 quit
```

A.4.7 Copy Different Types of OS/400 Files using forfiles and \$_file.type

Physical files are considered directories in UDM because they contain 1+ member. Save files are considered files because they do not contain any members. The **forfiles** statement and the variable **\$_file.type** allow you to do a wildcard copy on both save and physical files in the LIB file system.

This example copies a mix of files (Save and Physical) from an OS/400 system in a single operation, using the **forfiles** statement and the **\$_file.type** variable attribute.

```
forfiles src=MYLIB/*
if $_file.type EQ directory
copy src=$(_path)(*)
else
copy src=$(_path)
end
end
```

A.4.8 Invoke a Script from a Batch Job

To invoke a script included as an inline file in a database job, the call must specify ***FIRST** as the database member name.

The following example illustrates both:

- Invocation of an inline script, **CALLME**, using the STRUDM command from a database job.
- Invocation of an inline script, **CALL1**, using the CALL command from a database job.

LIB file system

```
//BCHJOB JOB(testcall) ENDSEV(10) OUTQ(mytest/UDMOUTQ) LOGCLPGM(*YES)
LOG(2 20 *SECLVL) MSGQ(*USRPRF)
//DATA FILE(CALL1) ENDCHAR(ENDDATAFILE)
print msg="I made it to call1 - an inline file"
ENDDATAFILE
//DATA FILE(CALLME) ENDCHAR(ENDDATAFILE)
OPEN S=AS400V5 USER=qatest PWD=***** PORT=4311
CALL CALL1(*FIRST)
CLOSE
ENDDATAFILE
STRUDM SCRFILE(CALLME)
//ENDBCHJOB
```

Appendix B

Customer Support

Stonebranch, Inc. provides customer support, via telephone and e-mail, for Universal Data Mover and all Universal Products.

TELEPHONE

Customer support via telephone is available 24 hours per day, 7 days per week.

North America

(+1) 678 366-7887, extension 6

(+1) 877 366-7887, extension 6 [toll-free]

Europe

+49 (0) 700 5566 7887

E-MAIL

All Locations

support@stonebranch.com

Customer support contact via e-mail also can be made via the Stonebranch website:

www.stonebranch.com



950 North Point Parkway, Suite 200
Alpharetta, Georgia 30005
U.S.A.

